Subcellular workflow

Release 1.0

João Pedro Gomes dos Santos, Kadri Pajo, Daniel Trpevski, Andr

2021/07/23

CONTENTS

(Under construction - last updated 2021/07/23)

This workflow has been developed to tackle the challenge of building and analyzing biochemical pathway models, combining pre-existing tools and custom-made software. (Santos et al. 2020) (Preprint)

At the root of our implementation is the SBtab format (Lubitz et al. 2016), a file that can store biochemical models and associated data in an easily readable and expandable way.

We have also developed tools to convert the SBtab format into several formats that can be used in MATLAB®, NEURON, STEPS and COPASI.

Using MATLAB® we have developed custom scripts for parameter estimation, and global sensitivities analysis, as well as diagnostics tools that can be used for model development. The global sensitivity analysis algorithm is modified from Halnes et al 2009.

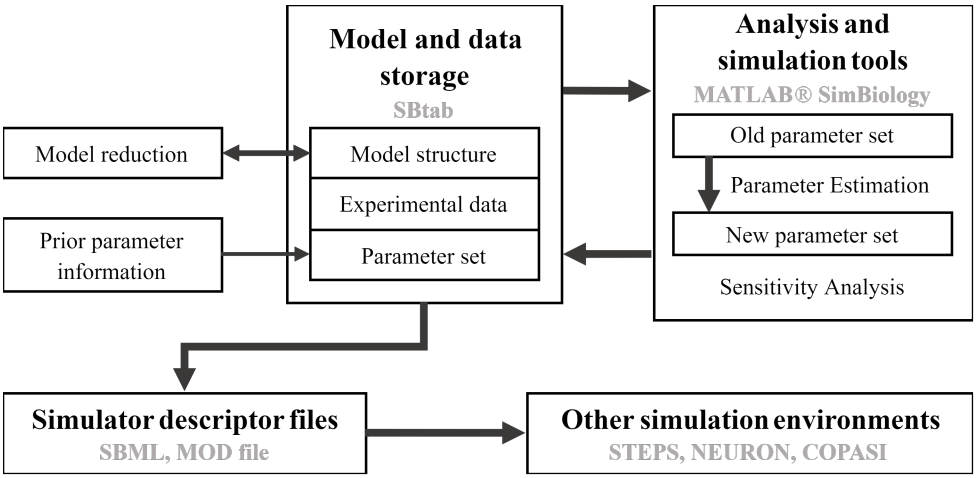
We demonstrate all these features using three example models, the main one being a modified version of the D1 MSN subcellular cascade model from Nair et al. 2016.

Code and files to run these models in different simulators:

- For MATLAB®
  Matlab/; Model_Nair_2016/Matlab; Model_Fujita_2010/Matlab; Model_Viswan_2018/Matlab
- For Neuron
  Model_Nair_2016; Model_Viswan_2018
- For the Subcellular application (STEPS)
  Model_Nair_2016/BioNetGen and STEPS/; Model_Viswan_2018/BioNetGen and STEPS/
- Copasi
  Model_Nair_2016; Model_Viswan_2018

Features:

- Wrapper for model simulation in MATLAB® (Matlab/Run_main.m)
- Analysis of selected parameter sets, using MATLAB® (Matlab/Run_main.m)
- Parameter optimization, using MATLAB® (Matlab/Run_main.m)
- Global Sensitivity analysis, using MATLAB® (Matlab/Run_main.m)
- Conversion tools:
  - SBtab (.xlsx,.xls) to SBtab (.tsv), using MATLAB® (Matlab/Run_main.m)
  - SBtab (.xlsx) to MATLAB® SimBiology® (.m, .sbproj), using MATLAB® (Matlab/Run_main.m)
  - MATLAB® SimBiology® to SBML (.xml), using MATLAB® (Matlab/Run_main.m)
    Needs to be fixed with our R script (https://github.com/a-kramer/simbiology-sbml-fix)
  - SBtab (.tsv) to VFGEN (.vf), using R (https://github.com/a-kramer/SBtabVFGEN)
  - SBtab (.tsv) to Mod (.mod), using R (https://github.com/a-kramer/SBtabVFGEN)
  - SBtab (.tsv) to SBML (.xml), using R (https://github.com/a-kramer/SBtabVFGEN)


Model and data storage
SBtab
Model structure
Experimental data
Parameter set
Model reduction
Prior parameter information
Analysis and simulation tools
MATLAB® SimBiology
Old parameter set
Parameter Estimation
New parameter set
Sensitivity Analysis
Simulator descriptor files
SBML, MOD file
Other simulation environments
STEPS, NEURON, COPASI

CHAPTER

ONE

SECTIONS OF THE WORKFLOW IN EXTERNAL REPOSITORIES

Conversion tools:

- https://github.com/a-kramer/SBtabVFGEN
- https://github.com/a-kramer/simbiology-sbml-fix

CHAPTER

TWO

IMPLEMENTED MODELS

- https://github.com/jpgsantos/Model_Nair_2016
- https://github.com/jpgsantos/Model_Fujita_2010
- https://github.com/jpgsantos/Model_Viswan_2018

CHAPTER

THREE

COMPATIBILITY

Subcellular workflow MATLAB® code is compatible with MATLAB® 2020a or above running on Microsoft Windows, macOS and Linux.

Matlab® packages needed:

- Optimization Toolbox™
- Statistics and Machine Learning Toolbox™
- Fuzzy Logic Toolbox™
- Financial Toolbox™
- Global Optimization Toolbox
- SimBiology®
- Parallel Computing Toolbox™

3.1 SBtab

This is an overview of the SBtab syntax that is used in our workflow. This contains a list of the sheet names and subfields that are read by our software. The second column in the first row of each sheet must include "TableName='sheet name'". Fields that are not mentioned here are not used in the latest workflow and are not imported but might be added to future releases. Additional information on how to use SBtab can be found in https://www.sbtab.net/sbtab/default/documentation.html.

3.1.1 Defaults

- **!ID** - Identifier code for the entries, it should consist of the letter "D" followed by an integer, should start at 1.
- **!Name** - Name of the default variable being defined, we have used time, volume, substance, lenght, and area in our files.
- **!Unit** - Unit of the default variable, eg. time - second.

3.1.2 Compartment

- **!ID** - Identifier code for the entries, it should consist of the letter "V" followed by an integer, should start at 1.
- **!Name** - Name of the compartment.
- **!Unit** - Compartment volume unit, usually liter.
- **!Size** - Size of the compartment in units defined in the unit column.

3.1.3 Compound

- **!ID** - Identifier code for the entries, it should consist of the letter "S" followed by an integer, should start at 0.
- **!Name** - Name of the compound. We advise to use recognizable compound names and separate complexes of multiple compounds with '_' (e.g. A_B). Must start with a letter.
- **!Unit** - Unit for the compound, usually nanomole or nanomole/liter. If it is not a default MATLAB |Reg| unit it should be added to it before trying to run the scripts.
- **!InitialValue** - Default initial values for the compounds before the equilibration step. These are usually overriden by the values *Si in the experiments sheet*
- **!IsConstant** - assigns a binary value, either TRUE or FALSE, depending on whether the value of a particular compound should stay constant throughout the simulations. Note that the input compound should remain constant.
- **!Assignment** - assigns a binary value, either TRUE or FALSE.
- **!Interpolation** -
- **!Type** - Type of the reaction, e.g. "kinetic".
- **!Location** - Compartment in which a particular compound is present.

3.1.4 Reaction

- **!ID** - Identifier code for the entries, it should consist of the letter "R" followed by an integer, should start at 0.
- **!Name** - Reaction name. Must start with a letter and it is advisable to include the reaction index, e.g. Reaction-Flux0.
- **!KineticLaw** - Reaction kinectic law. Needs to include the precise mass action kinetic law with compound and parameter names from corresponding SBtab table. For a reaction 'A + B <=> A_B' with a forward reaction rate of kf_R0 and backward reaction rate of kr_R0 the formula would look like 'kf_R0*A*B-kr_R0*A_B'.
- **!IsReversible** - Bollean identifying the reversibility of the reaction, it is TRUE for reversible and FALSE for irreversible reactions.
- **!Location** - Compartment in which a particular reaction is taking place.
- **!ReactionFormula** - Chemical formula of the reaction, should be written in the form 'A + B <= > A_B'.

3.1.5 Parameter

- **!ID** - Identifier code for the entries, it should consist of the letter "K" followed by an integer, should start at 0.
- **!Name** - Name of the parameter. We followed the convention of using 'kf' for forward reactions rates and 'kr' for reverese rates, followed by the *reaction !ID*, e.g. 'kf_R0'. These names can be arbitrary but they need to coincide with whatever is defined in the *Reaction kinectic law*
- **!Unit** - The units of the parameter.
- **!DefaultValue** - Parameter value in linear space.
- **!Value:linspace** - Parameter value in linear space.
- **!Value:log2** - Parameter value in log base 2.
- **!Value:log10** - Parameter value in log base 10.
- **!Location** - Compartment in which a particular parameter is governing a reaction.
- **!Comment** - Could be any plain text , we used it as a handy way of determining which reaction the parameter is involved in. We advise to use the following syntax 'kf_AXB__A_B' and 'kr_AXB__A_B' for respectively forward and backward rates of a reaction 'A + B <= > A_B'.

3.1.6 Expression

Compounds which are defined by expressions.

- **!ID** - Identifier code for the entries, it should consist of the letters "Ex" followed by an integer, should start at 0.
- **!Name** - Name of the compound defined by the expresion.
- **!Formula** - Formula assigned to the compound, it should use the compound names used in the compound sheet and, if needed, constant names from the constant sheet.
- **!Unit** - Concentration unit for the Compound.
- **!Location** - Compartment in which a particular compound is located.

3.1.7 Output

Compounds used as outputs in experimental data.

- **!ID** - Identifier code for the entries, it should consist of the letter "Y" followed by an integer, should start at 0.
- **!Name** - Name used to identify the output compound, when an existing compound needs to be measured we usually use "compound_name"_out.
- **!ErrorName** - Name of the error of the output compound. It should start with 'SD' (referring to standard deviation) followed by the output ID, e.g. SD_Y0.
- **!ErrorType** - Type of error for the output compound, we have used 'abs+rel random noise (std)'.
- **!Unit** - Concentration unit for the output compound.
- **!ProbDist** - Probability distribution type of the measured output in an experimental setting, e.g. 'normal'.
- **!Location** - Compartment in which a particular output compound is located.
- **!Formula** - Formula that links the experimental measured output to the compounds in the model. Usually the experimental measurement corresponds to a sum of compounds existing in the model but ratios are also common.

3.1.8 Experiments

Each column corresponds to one experiment for which there is a separate sheet.

- **!ID** - Identifier code for the entries, it should consist of the letter "E" followed by an integer, should start at 0.
- **!Name** - Name used to identify the experiment, we advise using the the word 'Experiment' followed by the experiment index.
- **>Output** - Should list all the output *!ID's*, i.e. Y's followed by their indices and separated by commas.
- **>S_i**- List of the various compounds of the model that have starting amounts other than 0, the same *!ID* as in the coumpound table should be used.

In a model with 2 experiments and 5 compounds A,B,C,D,E with IDs S0,S1,S2,S3,S4 respectively

- Experiment1 with compounds starting amounts A=0,B=1,C=2,D=0,E=3
- Experiment2 with compounds starting amounts A=1,B=0,C=1,D=0,E=4

Four entries should be included as exemplified bellow:

	>S0	>S1	>S2	>S4
Experiment1	0	1	2	3
Experiment2	1	0	1	4

Note that D/S3 is omitted because the starting value is 0 in all experiments.

- **!SimTime** - Simulation time for a particular experiment.
- **!Normalize** - Normalizations to be performed to the outputs are defined here.

3.1.9 E_i

Corresponds to individual experiments. It should have the name of the experiment IDs used in the experiments sheet.

- **!ID** - Identifier code for the entries, it should consist of the letters "E_iT" followed by an integer, should start at 0.
- **!Time** - Time series data, this should include a list of all the time points during which the corresponding output data points were sampled.
- **`>Y_i**- Compound to be measured, corresponds to the *!ID* of the output sheet. It should have the experimental (or simulated from another model) data.
- **`SD_Y_i**- Error of the compound to be measured, corresponds to the *!ErrorName* of the output sheet. It should have the experimental (or simulated from another model) data.

3.1.10 E_iI

Corresponds to individual experiments. It should have the name of the experiment IDs used in the experiments sheet, with an i in between E and the experiment number.

- **!ID** - Identifier code for the entries, it should consist of the letters "E_iIT" followed by an integer, should start at 0.
- **!Input_Time_S_i**- Time series of the inputs to the model, this should include a list of all the time points during which the corresponding input data points were sampled. To produce simple step inputs, only the time points during which a change in concentration is happening can be included. To produce more complicated input curves, more time points are needed to represent the shape of the curve.

- **`>S**$_i$- Compound that is being changed as input to the model, corresponds to the *!ID* in the compound table. This column should represent the sampled concentration data points corresponding to each time point.

3.2 MATLAB®

The MATLAB® section of this workflow has been developed to facilitate model building and rapid iteration between different versions of a model. In this workflow we use one main script, "Run_main.m", that calls all the relevant functions to be used. To run the MATLAB® code the Subcellular Workflow repository should be added to the MATLAB® path. Running the script "Run_main.m" generates prompts in the MATLAB® terminal window with a request to the user, to choose between a number of different options. These prompts allow the user to choose:

- The model to use (from all the models that are in the "Matlab/model" folder).

  Note that the very first time you run a model you have to add the folder for that specific model from its home repository into the "Matlab/model" folder(e.g. copy the folder "Model_nair_2016" from its repository to "Matlab/model/").

  For implemented models so far go to the following links:

  - Fujita_2010 model
  - Nair_2016 model
  - Viswan_2018 model

- The analysis to be performed, with the following options:

  1. *Diagnostics*
  2. *Parameter Estimation*
  3. *Global Sensitivity Analysis*
  4. Reproduction of a previous analysis

     This option can be used to re-do an analysis that has previously been performed. This is useful for reproducibility and in the case of the code getting updated with extra funcionalities. The user should specify the analysis file that they want to use, examples are provided in the each model repository.

  5. Reproduction of the plots of a previous analyis

     Similar to the previous option but here only the plots are re-done.

  6. Import model files

     Creation of the model files and folder that are needed to run the model in Matlab, the creation of a folder with the model in .tsv format (one tsv file for each excel sheet of the original SBtab), as well as the conversion of the model to the SBML format (.xml).

- The *settings file* to use on the model.
  These settings files can be found either in the respective model repository in the directory "Matlab/Settings", or in the example model from our main repository in the directory "Matlab/model/Model_Example/Matlab/Settings", or by following these links:

  - Example model settings files
  - Fujita_2010 model settings files
  - Nair_2016 model settings files
  - Viswan_2018 model settings files

Examples of the output recieved when the different models are run through the workflow can be found on the respective model repository in the directory "Matlab/Results/Results/Examples", or by following these links:

- Fujita_2010 model example results
- Nair_2016 model example results
- Viswan_2018 model example results

In order to gain a better understanding on how the code works, there are detailed pages for the following:

- *Scripts* - The script that we use;
- *Functions* - All the custom functions we have built; this is directed to anyone that wants to develop or iterate the code;
- *Settings file* - The master configuration file, where we describe everything that can be modified by the user without changing any code;
- *Results* - Explanation of all the files containing relevant results that are generated after running the built-in analysis of the code;
- *Model files and folders* - Description of all the files and folders that are generated when the model is imported from SBtab into relevant files, used by the rest of the MATLAB® code.

3.2.1 Diagnostics

The specifications of the diagnostics operations require user input in the *settings file*. The toolkit imports the model stored in *SBtab format* (with the name specified by *stg.sbtab_excel_name* variable from a folder chosen by the user terminal prompts) to a .sbproj file, and saves it in the subfolder called Data along with the imported data and inputs in a .mat format. Once the model has been imported, the import function can be disabled in the *import section of the settings file* for further procedures.

The parameter sets that are specified in the *diagnostics section of the settings file* are then used in model simulations with the input specifications (e.g. time series data for relevant input species) in the Experiment Input (EI) tables in the SBtab file, calculate the error between the simulation results and the experimental data, and plot the error scores as well as the comparative traces of the simulation results and the experimental data. At least one parameter set is currently required in the settings file. Experiments of interest can be specified by the *stg.exprun* variable in the *analysis section of the settings file*.

3.2.2 Parameter Estimation

Parameter estimation is performed by various MATLAB optimization algorithms. The number of parameters to estimate, possible thermodynamic constraints (can be determined by a standalone script), and upper and lower bounds can be specified in the *model section of the settings file*. In addition, the parameter indices and the best available parameter sets can specified in the *diagnostics section*. Optimization algorithm and optimization settings can be found in the *optimziation section of the settings file* and simulation settings in the *simulation section*.

3.2.3 Global Sensitivity Analysis

Global Sensitivity Analysis can be performed when a user is interested in parameter distribution rather than single point values, and how sensitive a specific output is towards perturbations in different parameters. Instructions on what settings are required to be specified can be found in the *Global Sensitivity Analysis section of the settings file*.

3.2.4 Scripts

This is the entry point for our code, it calls all other relevant functions.

Run_main

Code

```
% Script to import sbtab and run the analyis
% clear functions

%Get the date and time
date_stamp = string(year(datetime)) + "_" + ...
    string(month(datetime)) + "_" + string(day(datetime))...
    + "__" + string(hour(datetime)) + "_" + string(minute(datetime))...
    + "_" + string(round(second(datetime)));

% Get the folder where the MATLAB code and models are located
Matlab_main_folder = fileparts(mfilename('fullpath'))+"/";
Matlab_main_folder = strrep(Matlab_main_folder,"\","/");
addpath(genpath(Matlab_main_folder));
mmf.main = Matlab_main_folder;

% Name of the various analysis that can be run with this code
analysis_options = ["Diagnostics","Parameter Estimation",...
    "Global Sensitivity Analysis","Reproduce a previous analysis",...
    "Reproduce the plots of a previous analysis","Import model files"];

% Code for choosing the model and loading the settings files
[stg,rst,sb] = f_user_input(mmf,analysis_options);

% Get the folder structure used for the model files
[mmf] = default_folders(stg,mmf,date_stamp);

% Runs the import scripts if chosen in settings
if stg.import
    [stg,sb] = f_import(stg,mmf);
else
    % Creates a struct based on the sbtab that is used elswhere in the code
    % and also adds the number of experiments and outputs to the settings
    % variable
    if isempty(sb)% check needed for plot reproduction
        [stg,sb] = f_generate_sbtab_struct(stg,mmf);
    end
end

```

(continues on next page)

(continued from previous page)

```
% Runs the Analysis chosen in settings
if any(contains(analysis_options(1:3),stg.analysis))
    rst = f_analysis(stg,stg.analysis,mmf,analysis_options);
end
% Save Analysis results if chosen in settings
if stg.save_results
    f_save_analysis(stg,sb,rst,mmf)
end

% Plots the results of the analysis, this can be done independently after
% loading the results of a previously run analysis
if stg.plot
    f_plot(rst,stg,mmf)
    % Save plots results if chosen in settings
    if stg.save_results
        f_save_plots(mmf)
    end
end
```

This is the main script from the MATLAB® portion of the workflow. Depending on the configurations on the *settings file* and choices on the user facing prompts it can call functions to:

- *Perform conversions of the SBtab:*
  - SBtab(.xlsx) to SBtab (.tsv)
  - SBtab (.xlsx) to MATLAB® SimBiology® (.m, .sbproj)
  - MATLAB® SimBiology® to SBML (.xml)
- *Perform analysis on the model:*
  - Diagnostics
  - Parameter Estimation
  - Global Sensitivity Analysis
- *Saving results from analysis*
- *Plotting relevant results*
- *Saving plots*

It can also reproduce a the calculations of a previous analysis or just its plots.

3.2.5 Functions

The MATLAB® functions used in this workflow are divided acording to their role, we have:

- *Setup and Import functions*

  Functions to import the model to MATLAB® and generate model specific files and functions.

- *Simulation and scoring functions*

  Functions relating to the simulation and scoring of the model.

- *Analysis functions*

  Functions for the analysis that we can perform on the model.

- *Ploting unctions*

  Functions to plot the result of the different analyses.

- *General purpose functions*

  General purpose functions that are usually used by other functions.

Setup and Import

f_load_settings

Code

```
function [stg,rst,sb] = f_user_input(mmf,analysis_options)

persistent last_SBtab_date
persistent last_model_folder
persistent last_settings_file_text
persistent last_settings_file_date
persistent last_analysis_text

Matlab_main_folder = mmf.main;

rst = [];
sb = [];
functions_cleared = false;

% Get the folder of the model
model_folder_general = Matlab_main_folder + "Model/";
last_choice = last_model_folder;
prompt = "What model folder should be used?\n";
[model_folder,last_model_folder] =...
    choose_options(model_folder_general,prompt,last_choice);

model_name_specific = string(model_folder);
folder_model_specific = model_folder_general + "/" + model_name_specific;

prompt = "\nWhat analysis should be performed?\n";
last_choice = last_analysis_text;
[analysis_text,last_analysis_text] =...
    parse_choices(prompt,analysis_options,last_choice);

% analysis_n = find(contains(analysis_options,analysis_text));

% Check if an analysis was chosen
if any(contains(analysis_options([1:3,6]),...
        analysis_text))

    %Get the Setting file to be used
    settings_folder = folder_model_specific + "/Matlab/Settings";
    last_choice = last_settings_file_text;
    prompt = "\nWhat file should be used as settings?\n";
    [settings_file_text,last_settings_file_text] =...
```

(continues on next page)

(continued from previous page)

```
        choose_options(settings_folder,prompt,last_choice);

    settings_file = strrep(settings_file_text,".m","");

    % Add the default settings to the struct
    stg_add_default = eval("default_settings()");

    f = fieldnames(stg_add_default);
    for i = 1:length(f)
        stg.(f{i}) = stg_add_default.(f{i});
    end

    % Add chosen settings to the struct overwriting defaults when
    % appropriate
    [stg_add] = eval(settings_file + "()");

    f = fieldnames(stg_add);
    for i = 1:length(f)
        stg.(f{i}) = stg_add.(f{i});
    end

    % Check if the date of the settings file changed, if so clear functions
    listing = dir(settings_folder);
    for n = 1:size(listing,1)
        if matches(settings_file_text,listing(n).name,"IgnoreCase",true)
            settings_file_date = listing(n).date;
        end
    end

    [last_settings_file_date,functions_cleared] =...
        compare_last(settings_file_date,last_settings_file_date,...
        functions_cleared);

    % Check if the name of the settings file changed, if so clear functions
    [last_settings_file_text,functions_cleared] =...
        compare_last(settings_file_text,last_settings_file_text,...
        functions_cleared);

    % Check if the date of the SBtab changed, if so clear functions
    listing = dir(folder_model_specific);

    for n = 1:size(listing,1)
        if matches(stg.sbtab_excel_name,listing(n).name,"IgnoreCase",true)
            sbtab_date = listing(n).date;
        end
    end
    [last_SBtab_date,~] =...
        compare_last(sbtab_date,last_SBtab_date,functions_cleared);

    % Store the name of the chosen analysis in the settings struct
    stg.analysis = analysis_text;

```

(continues on next page)

(continued from previous page)

```
    if contains(analysis_options(6),analysis_text)
        stg.import = true;
        stg.save_results = false;
        stg.plot = false;
    end
elseif any(contains(analysis_options(4:5),analysis_text))

    % Get the folder of the Analysis that should be reproduced
    folder_results = folder_model_specific + "/Matlab/Results";

    last_choice = [];
    prompt = "\nWhat analysis should be reproduced?\n";

    [r_analysis_text,~] =...
        choose_options(folder_results,prompt,last_choice);

    folder_results_specific = folder_results + "/" + ...
        r_analysis_text;

    last_choice = [];
    prompt = "\nWhen was this analysis run originaly?\n";

    [r_analysis_date_text,~] =...
        choose_options(folder_results_specific,prompt,last_choice);

    folder_results_specific_date = folder_results_specific + "/" + ...
        r_analysis_date_text;

    % Load the settings file and the SBtab struct
    load(folder_results_specific_date + "/Analysis.mat","stg","sb")

    % Set inport to false since we don't want to overwrite anything
    stg.import = false;

    % If the reproduction of an analysis is chosen clear the functions
    % because the settings most likely changed
    if contains(analysis_options(4),analysis_text)
        f_functions_to_clear()
    end

    % I the reproduction of the plots of an analyis is chosen make sure
    % we tell the code to produce plots and also load the results that were
    % previously obtained
    if contains(analysis_options(5),analysis_text)
        stg.plot = true;
        load(folder_results_specific_date + "/Analysis.mat","rst")
    end
end

% Set the chosen model folder in the settings struct
stg.folder_model = model_name_specific;
end
```

(continues on next page)

(continued from previous page)

```

function [choice,last_choice] = choose_options(folder,prompt,last_choice)

listing = dir(folder);

for n = size(listing,1):-1:1
    if any(matches(listing(n).name,[".","..","Place models here.txt"]))
        listing(n)= [];
    end
end

for n = 1:size(listing,1)
    options(n) = string(listing(n).name);
end

[choice,last_choice] = parse_choices(prompt,options,last_choice);
end

function [choice,last_choice] = parse_choices(prompt,options,last_choice)

for n = 1:size(options,2)
    prompt = prompt + "\n" + n + ": " + options(n);
end

if ~isempty(last_choice)
    if any(contains(options,last_choice))
        prompt = prompt + "\n\nPress enter to use " + last_choice;
    else
        last_choice = [];
    end
end

prompt = prompt + "\n";

i = input(prompt);

if isempty(i)
    choice = [];
elseif i > 0 && i < size(options,2)+1
    choice = options(i);
    disp("The option chosen was: " + choice)
else
    prompt = "Please choose from the provided options";
    [choice,last_choice] = parse_choices(prompt,options,last_choice);
end

if isempty(choice)
    if ~isempty(last_choice)
        choice = last_choice;
        disp("The option chosen was: " + last_choice)
    else
        prompt = "Please choose from the provided options";
```

(continues on next page)

(continued from previous page)

```
        [choice,last_choice] = parse_choices(prompt,options,last_choice);
    end
else
    last_choice = choice;
end
end

function [previous,f_cleared] = compare_last(current,previous,f_cleared)

if ~isempty(previous)
    if ~contains(current,previous)
        if f_cleared == false
            disp("Settings file changed, clearing functions")
            f_functions_to_clear()
            f_cleared = true;
        end
    end
end
previous = current;
end
```

It prompts the user to choose the model to run, the settings file to use, and the Analysis to perform.

- **Outputs** - *stg*, *rst*, sb, Analysis_n

f_import

Code

```
function [stg,sb] = f_import(stg,mmf)

Model_folder = mmf.model.main;

disp("Generating model files and folder from SBtab")

% Create needed folders
[~,~] = mkdir(mmf.model.data.main);
[~,~] = mkdir(mmf.model.input_functions.main);
[~,~] = mkdir(mmf.model.tsv.model_name);
[~,~] = mkdir(mmf.model.data.model_exp.main);

% Creates a .mat and a tsvs from the SBtab file
f_excel_sbtab_importer(mmf);

addpath(genpath(Model_folder));

% Creates a struct based on the SBtab that is used elswhere in the code and
% also adds the number of experiments and outputs to the settings struct
[stg,sb] = f_generate_sbtab_struct(stg,mmf);

%Create the model and input output structure from sbtab.mat

```

(continues on next page)

(continued from previous page)

```
% Saves the model in .mat, .sbproj and .xml format, while also creating a
% file whith the data to run the model in all different experimental
% settings defined in the SBtab
f_sbtab_to_model(stg,sb,mmf)

% Creates code that loads the inputs of each experiment into a .mat file,
% and creates the code to read this inputs at runtime when the experiments
% are being simulated, all this generated code is stored on the Input
% functions folder
f_setup_input(stg,mmf)

%Creates three .mat files for each experiment, with all the added rules,
%species and parameters needed depending on the inputs and outputs
%specified on the SBtab, one for the equilibrium simulation run, one for
%the deffault run, and one for a more detailed run.
f_build_model_exp(stg,sb,mmf)
disp("Model files and folders generated successfully")
end
```

Creates the necessary folders inside the model folder. Calls subfunctions that convert the SBtab from an Excel into MATLAB® files useful for the workflow, TSVs and a SBML.

f_excel_sbtab_importer

Code

```
function f_excel_sbtab_importer(mmf)

Source_sbtab = mmf.model.sbtab;
Matlab_sbtab = mmf.model.data.sbtab;

% Get the total number of sheets in the SBTAB
sheets = sheetnames(Source_sbtab);

% Try to run the import the sheets in multicore, depending on the version
% of excel this migth not work
try
    parfor i = 1:size(sheets,1)
        sbtab_excel{i} = impexp (i,mmf);
    end
catch
    for i = 1:size(sheets,1)
        sbtab_excel{i} = impexp (i,mmf);
    end
end

% Save the SBTAB tables in .mat format
save(Matlab_sbtab,'sbtab_excel');
disp("SBtab with " + size(sheets,1) + " sheets parsed successfully")
end

```

(continues on next page)

(continued from previous page)

```
function sbtab_excel = impexp (i,mmf)

Source_sbtab = mmf.model.sbtab;
tsv_name_folder = mmf.model.tsv.model_name;

% Import the SBTAB to a cell with a sheet per cell
sbtab_excel = readcell(Source_sbtab,'sheet',i);

% Replace "ismissing" values with empty spaces
mask = cellfun(@ismissing, sbtab_excel,'UniformOutput',false);
mask = cellfun(@min, mask);
mask = logical(mask);
sbtab_excel(mask) = {[]};

% Get name for tsv that is going to be exported
field = regexp(sbtab_excel{1,2},"TableName='[^']*'",'match');
field = string(replace(field,["TableName='","'"," "],["","","_"]));

%Export the tsv
cell_write_tsv(tsv_name_folder + field + ".tsv",sbtab_excel)
end

function cell_write_tsv(filename,origCell)

% save a new version of the cell for reference
modCell = origCell;
% assume some cells are numeric, in which case set to char
iNum = cellfun(@isnumeric,origCell);

% Replace numeric sells with cell strings
for n = 1:size(iNum,1)
    for m = 1:size(iNum,2)
        modCell(n,m) = cellstr(num2str(origCell{n,m}));
    end
end

%Save the file that only as strings in each cell
modCell = transpose(modCell);

[rNum,cNum] = size(origCell);
frmt = repmat([repmat('%s\t',1,cNum-1),'%s\n'],1,rNum);
fid = fopen(filename,'wt');
fprintf(fid,frmt,modCell{:});
fclose(fid);
end
```

Loads the information in the SBtab and creates a . *mat file* that contains the sbtab and *TSVs* corresponding to all the SBtab tabs.

- **Inputs** - *stg*
- **Saves**
  - *.mat file containing the SBtab* in the "Model/Data" folder

- *TSVs containing the SBtab* in the "Model/tsv" folder

f_generate_sbtab_struct

Code

```
function [stg,sb] = f_generate_sbtab_struct(stg,mmf)

Matlab_sbtab = mmf.model.data.sbtab;

if isfile(Matlab_sbtab)

    load(Matlab_sbtab,'sbtab_excel');

    sb = f_get_sbtab_fields(sbtab_excel);

    stg.expn = size(sb.Experiments.ID,1);
    stg.outn = size(sb.Output.ID,1);
end
end

function sb = f_get_sbtab_fields(sbtab_excel)
for n = 1:size(sbtab_excel,2)

    if ~isempty(sbtab_excel{1,n}{1,2})

        field = regexp(sbtab_excel{1,n}{1,2},"TableName='[^']*'",'match');
        field = string(replace(field,["TableName='","'"," "],["","","_"]));

        for k = 1:size(sbtab_excel{1,n},2)

            if ~isempty(sbtab_excel{1,n}{2,k})
                subfield = sbtab_excel{1,n}{2,k};
                subfield =...
                    string(replace(subfield,["!",">",":"," "],["","","_","_
↪"]));

                sb.(field).(subfield)(:,1) = sbtab_excel{1,n}(3:end,k)';

                sb.(field).(subfield) = sb.(field).(subfield)...
                    (~cellfun('isempty', sb.(field).(subfield)));
            end
        end
    end
end
end
```

Loads the SBtab saved in the *.mat file* and creates a MATLAB® struct that can be more easily parsed.

- **Inputs** - *stg*
- **Outputs** - sb, *stg.expn*, *stg.outn*.

f_sbtab_to_model

Code

```
function f_sbtab_to_model(stg,sb,mmf)
% Saves the model in .mat, .sbproj and .xml format, while also creating a
% file whith the data to run the model in all different experimental
% settings defined in the sbtab

modelobj = sbiomodel(stg.name);
compObj = [];

sbtab.species = cat(2,sb.Compound.Name,sb.Compound.InitialValue,...
    sb.Compound.IsConstant,sb.Compound.Unit,sb.Compound.Location);

sbtab.defpar = cat(2,sb.Parameter.Comment,sb.Parameter.Value_linspace,...
    sb.Parameter.Unit);

for n = 1:size(sb.Compartment.ID,2)
    compObj{n} = addcompartment(modelobj, sb.Compartment.Name{n});
    set(compObj{n}, 'CapacityUnits', sb.Compartment.Unit{n});
    set(compObj{n}, 'Value', sb.Compartment.Size{n});
end

for n = 1:size(sbtab.species,1)

    for m = 1:size(compObj,2)
        if string(compObj{m}.Name) == string(sb.Compound.Location{n})
            compartment_number_match = m;
        end
    end

    addspecies (compObj{compartment_number_match}, sb.Compound.Name{n},sb.Compound.InitialValue{n}...
        ,'InitialAmountUnits',sb.Compound.Unit{n});
end

for n = 1:size(sbtab.defpar,1)
    addparameter(modelobj,sb.Parameter.Name{n},...
        sb.Parameter.Value_linspace{n},'ValueUnits',sb.Parameter.Unit{n},'Notes',sb.Parameter.Comment{n});
end

for n = 1:size(sb.Reaction.ID,1)

    if ischar(sb.Reaction.IsReversible{n})
        if contains(convertCharsToStrings(sb.Reaction.IsReversible{n}),"true")
            reaction_name = strrep(sb.Reaction.ReactionFormula{n},'<=>',' <-> ');
        else
```

(continues on next page)

(continued from previous page)

```
            reaction_name = strrep(sb.Reaction.ReactionFormula{n},'<=>
↪',' -> ');
        end
    else
        if sb.Reaction.IsReversible{n}
            reaction_name = strrep(sb.Reaction.ReactionFormula{n},'<=>
↪',' <-> ');
        else
            reaction_name = strrep(sb.Reaction.ReactionFormula{n},'<=>
↪',' -> ');
        end
    end

    reaction_name_compartment = reaction_name;

    for m = 1:size(sb.Compound.Name,1)
        reaction_name_compartment =...
            insertBefore(string(reaction_name_compartment)," " +...
            string(sb.Compound.Name{m})," " + string(sb.Reaction.
↪Location{n}));
    end

    while contains(reaction_name_compartment,...
            string(sb.Reaction.Location{n})+" "+string(sb.Reaction.
↪Location{n}))
        reaction_name_compartment =...
            strrep(reaction_name_compartment,string(sb.Reaction.
↪Location{n})+...
            " "+string(sb.Reaction.Location{n})," "+sb.Reaction.
↪Location{n});
    end

    while contains(reaction_name_compartment,"  ")
        reaction_name_compartment = strrep(reaction_name_compartment,"␣
↪ "," ");
    end

    reaction_name_compartment = strrep(reaction_name_compartment,...
        sb.Reaction.Location{n} + " ",sb.Reaction.Location{n}+".");
    reaction_name_compartment = string(sb.Reaction.Location{n})+".
↪"+reaction_name_compartment;

    reactionObj = addreaction(modelobj,reaction_name_compartment);
    set(reactionObj,'ReactionRate',sb.Reaction.KineticLaw{n});
end

for n = 1:size(sb.Compound.ID,1)
    if ischar(sb.Compound.Assignment{n})
        if contains(convertCharsToStrings(sb.Compound.Assignment{n}),[
↪"true","True"])
            modelobj.species(n).BoundaryCondition = 1;
        end
```

(continues on next page)

(continued from previous page)

```
    else
        if sb.Compound.Assignement{n} == 1
            modelobj.species(n).BoundaryCondition = 1;
        end
    end
    if ischar(sb.Compound.Interpolation{n})
        if contains(convertCharsToStrings(sb.Compound.Interpolation{n}
↪),["true","True"])
            modelobj.species(n).BoundaryCondition = 1;
        end
    else
        if sb.Compound.Interpolation{n} == 1
            modelobj.species(n).BoundaryCondition = 1;
        end
    end
    if ischar(sb.Compound.IsConstant{n})
        if contains(convertCharsToStrings(sb.Compound.IsConstant{n}),[
↪"true","True"])
            modelobj.species(n).BoundaryCondition = 1;
        end
    else
        if sb.Compound.IsConstant{n} == 1
            modelobj.species(n).BoundaryCondition = 1;
        end
    end
end

sbtab.sim_time = [sb.Experiments.Sim_Time{:}];

species_INP_matcher ={};
for n = 1:size(sb.Compound.ID,1)
    if isfield(sb.Experiments,"S"+(n-1))
        species_INP_matcher{size(species_INP_matcher,1)+1,1} = n;
    end
end

for n = 1:size(sb.Experiments.ID,1)
    startamount = cell(1,size(species_INP_matcher,1));
    nstartamount = 0;
    nInputTime = 0;
    nInput = 0;
    nOutput = 0;

    for m = 1:size(sb.Compound.ID,1)
        if isfield(sb.Experiments,"S"+(m-1))
            nstartamount = nstartamount+1;
            startamount{nstartamount} = eval("sb.Experiments.S"+(m-1)+
↪"(n)");
            startAmountName(nstartamount) = sb.Compound.Name(m);
        end
    end

```

(continues on next page)

(continued from previous page)

```
    if isfield(sb.Experiments,"Normalize")
        sbtab.datasets(n).Normalize = sb.Experiments.Normalize{n};
    else
        sbtab.datasets(n).Normalize = [];
    end

    if isfield(eval(("sb.E")+(n-1)),"Time")
        Data(n).Experiment.t = transpose(eval("[sb.E"+(n-1)+".Time{:}]
↪"));
    end

    for m = 1:size(sb.Compound.ID,1)
        if isfield(eval(("sb.E")+(n-1)+"I"),"Input_Time_S"+(m-1))
            nInputTime = nInputTime + 1;
            sbtab.datasets(n).input_time{1,nInputTime} = ...
                eval(("[sb.E") + (n-1) + "I.Input_Time_S" + (m-1) + "
↪{:}]");
        end
        if isfield(eval(("sb.E")+(n-1)+"I"),"S"+(m-1))
            nInput = nInput + 1;
            sbtab.datasets(n).input_value{1,nInput} = ...
                eval(("[sb.E") + (n-1) + "I.S" + (m-1) + "{:}]");
            sbtab.datasets(n).input{nInput} = char("S" + (m-1));
        end
    end

    for m = 1:size(sb.Output.ID,1)
        if isfield(eval(("sb.E")+(n-1)),"Y"+(m-1))
            nOutput = nOutput+1;
            Data(n).Experiment.x(:,nOutput) = eval(("[sb.E") +...
                (n-1) + ".Y" + (m-1) + "{:}]");
            Data(n).Experiment.x_SD(:,nOutput) = eval(("[sb.E") +...
                (n-1) + ".SD_Y" + (m-1) + "{:}]");
            sbtab.datasets(n).output{nOutput} = sb.Output.Name(m);
            sbtab.datasets(n).output_value{nOutput} = ...
                {convertStringsToChars(...
                strrep(string(sb.Output.Location{m}) + "." +...
                string(sb.Output.Name{m}) + " = " +...
                string(sb.Output.Formula{m}),'eps','0.0001'))};
            sbtab.datasets(n).output_name{nOutput} = ...
                sb.Output.Name(m);
            sbtab.datasets(n).output_ID{nOutput} = ...
                sb.Output.ID(m);
            sbtab.datasets(n).output_location{nOutput} = ...
                sb.Output.Location(m);
        end
    end
    sbtab.datasets(n).stg.outnumber = nOutput;
    sbtab.datasets(n).start_amount = cat(2,startAmountName(:)...
        ,transpose([startamount{:}]),species_INP_matcher);
end

```

(continues on next page)

(continued from previous page)

```
if isfield(sb,"Expression")
    for m = 1:size(sb.Expression.ID,1)
        if isfield(sb.Expression,'Formula')
            if isa(sb.Expression.Formula{m},'double')
                addspecies (modelobj, char(sb.Expression.Name(m)),...
                    str2double(string(sb.Expression.Formula{m})),...
                    'InitialAmountUnits',sb.Expression.Unit{m});
            else
                try
                    addspecies (modelobj, char(sb.Expression.Name(m)),
↪0,...
                        'InitialAmountUnits',sb.Expression.Unit{m});
                catch
                end
                addrule(modelobj, char({convertStringsToChars(...
                    string(sb.Expression.Location{m}) + "." +...
                    string(sb.Expression.Name{m}) + " = " +...
                    string(sb.Expression.Formula{m}))}),...
                    'repeatedAssignment');
            end
        else
            addparameter(modelobj,char(sb.Expression.Name(m)),...
                str2double(string(sb.Expression.DefaultValue{m})),...
                'ValueUnits',sb.Expression.Unit{m});
        end
    end
end

if isfield(sb,"Input")
    for m = 1:size(sb.Input.ID,1)
        if isfield(sb.Input,'Formula')
            if isa(sb.Input.Formula{m},'double')
                addspecies (modelobj, char(sb.Input.Name(m)),...
                    str2double(string(sb.Input.DefaultValue{m})),...
                    'InitialAmountUnits',sb.Input.Unit{m});
            else
                try
                    addspecies (modelobj, char(sb.Input.Name(m)),0,...
                        'InitialAmountUnits',sb.Input.Unit{m});
                catch
                end
                addrule(modelobj, char({convertStringsToChars(...
                    string(sb.Input.Location{m}) + "." +...
                    string(sb.Input.Name{m}) + " = " +...
                    string(sb.Input.DefaultValue{m}))}),...
                    'repeatedAssignment');
            end
        else
            addparameter(modelobj,char(sb.Input.Name(m)),...
                str2double(string(sb.Input.DefaultValue{m})),...
                'ValueUnits',sb.Input.Unit{m});
        end
```

(continues on next page)

(continued from previous page)

```
    end
end

if isfield(sb,"Constant")
    for m = 1:size(sb.Constant.ID,1)
        addparameter(modelobj,char(sb.Constant.Name(m)),...
            str2double(string(sb.Constant.Value{m})),...
            'ValueUnits',sb.Constant.Unit{m});
    end
end

sbproj_model = mmf.model.data.sbproj_model;
matlab_model = mmf.model.data.mat_model;
data_model = mmf.model.data.data_model;
xml_model = mmf.model.data.xml_model;

sbiosaveproject(sbproj_model,'modelobj')

save(matlab_model,'modelobj')

save(data_model,'Data','sbtab','sb')

sbmlexport(modelobj,xml_model)

end
```

Reads information from the SBtab and saves the model in MATLAB (*.mat*, *.sbproj*) and SBML(*.xml*) format, while also creating a *file* whith the data to run the model in all different experimental settings defined in the SBtab.

- **Inputs** - *stg*, sb
- **Saves** - *model file .mat*, *model file .sbproj*, *model file .xml*, and *data file*

f_setup_input

Code

```
function f_setup_input(stg,mmf)
% Creates code that loads the inputs of each experiment into a .mat
↪file,
% and creates the code to read this inputs at runtime when the
↪experiments
% are being simulated, all this generated code is stored on the
% Input_functions folder

matlab_model = mmf.model.data.mat_model;
data_model = mmf.model.data.data_model;
inp_model_data = mmf.model.data.input_model_data;
Model_folder = mmf.model.main;
model_input = mmf.model.input_functions.input;

%Find correct path for loading depending on the platform
```

(continues on next page)

(continued from previous page)

```
load(data_model,'sbtab')

load(matlab_model,'modelobj');

for Exp_n = 1:size(sbtab.datasets,2)

    for index = 1:size(sbtab.datasets(Exp_n).input,2)
        if size(sbtab.datasets(Exp_n).input_value{index},2) > 100

            input_name = strrep(modelobj.species(1+str2double(strrep(..
↪.
                sbtab.datasets(Exp_n).input(index),'S',''))).name,".","
↪");
            inpX = input_name + "X";
            inpT = input_name + "T";
            inph1 = input_name + "h1";
            inph2 = input_name + "h2";

            fullFileName = sprintf('%s.m',...
                model_input + Exp_n + "_" + input_name  );

            fileID = fopen(fullFileName, 'wt');

            inp_str = template1();
            inp_str = replace(inp_str,...
                ["SBtab_name","Exp_n","input_name",...
                "inpX", "inpT", "inph1", "inph2", "inp_model_data",...
                "sbtab.sim_time(Exp_n)"],[stg.name,Exp_n,...
                input_name, inpX, inpT, inph1,...
                inph2, inp_model_data,...
                sbtab.sim_time(Exp_n)]);

            fprintf(fileID,inp_str);
            fclose(fileID);
        end
    end
end

fullFileName = sprintf('%s.m',model_input + "_creator"  );

fileID = fopen(fullFileName, 'wt');
fprintf(fileID, "function " + stg.name + "_input_creator(~)\n");
helper = 0;
for Exp_n = 1:size(sbtab.datasets,2)
    for index =1:size(sbtab.datasets(Exp_n).input,2)
        if size(sbtab.datasets(Exp_n).input_value{index},2) > 100
            input_name = strrep(modelobj.species(1+str2double(strrep(..
↪.
                sbtab.datasets(Exp_n).input(index),'S',''))).name,".","
↪");
            if helper == 0
                helper = 1;
```

(continues on next page)

(continued from previous page)

```
                helper2 = Exp_n;
            end
            if index == 1 && Exp_n == helper2
                fprintf(fileID,"load('" + data_model + "','sbtab');\n
↪");
                inp_creator_str = template2();
                inp_creator_str = replace(inp_creator_str,...
                    ["Exp_n", "input_name", "index", "inp_model_data"],
↪...
                    [Exp_n, input_name, index, inp_model_data]);
                %                 fprintf(fileID,inp_creator_str);
            else
                inp_creator_str = template3();
                inp_creator_str = replace(inp_creator_str,...
                    ["Exp_n", "input_name", "index", "inp_model_data"],
↪...
                    [Exp_n, input_name, index, inp_model_data]);
            end
            fprintf(fileID,inp_creator_str);
        end
    end
end
fprintf(fileID, "end\n");
fclose(fileID);

addpath(genpath(Model_folder));
eval(stg.name + "_input_creator()");
end

function inp_str = template1()
inp_str =...
    "function thisAmp = SBtab_name_inputExp_n_input_name(times)\n"+...
    "persistent inpX\n"+...
    "persistent inpT\n"+...
    "persistent inph1\n"+...
    "persistent inph2\n"+...
    "if isempty(inpX)\n"+...
        "Data = coder.load('inp_model_data','expExp_n_input_name');\n
↪"+...
        "inpX = Data.expExp_n_input_name(:,2);\n"+...
        "inpT = Data.expExp_n_input_name(:,1);\n"+...
        "inph1 = 1;\n"+...
    "end\n"+...
    "thisAmp = inpX(end);\n"+...
    "if times ~= sbtab.sim_time(Exp_n)\n"+...
        "if times == 0\n"+...
            "inph1 = 1;\n"+...
            "thisAmp = inpX(1);\n"+...
            "else\n"+...
            "while times > inpT(inph1)\n"+...
            "inph1 = inph1 + 1;\n"+...
        "end\n"+...
```

(continues on next page)

(continued from previous page)

```
        "while times < inpT(inph1-1)\n"+...
            "inph1  = inph1-1;\n"+...
            "end\n"+...
            "inph2 = (inpT(inph1)-times)*1/(inpT(inph1)-inpT(inph1-1));\n"+...
            "thisAmp = (inpX(inph1-1)*inph2 + inpX(inph1)*(1-inph2));\n"+...
        "end\n"+...
    "end\n"+...
    "end";
end


function inp_creator_str = template2()
inp_creator_str = ...
    "expExp_n_input_name(:,1) = sbtab.datasets(Exp_n).input_time{index};\n"+...
    "expExp_n_input_name(:,2) = sbtab.datasets(Exp_n).input_value{index};\n"+...
    "save('inp_model_data','expExp_n_input_name');\n";
end
function inp_creator_str = template3()
inp_creator_str = ...
    "expExp_n_input_name(:,1) = sbtab.datasets(Exp_n).input_time{index};\n"+...
    "expExp_n_input_name(:,2) = sbtab.datasets(Exp_n).input_value{index};\n"+...
    "save('inp_model_data','expExp_n_input_name','-append');\n";
end
```

Creates code that loads the inputs of each experiment into a *.mat file* and the code to read these inputs at runtime when the experiments are being simulated. All this generated code is stored on the "Model/«model folder name»/Formulas" folder.

- **Inputs** - *stg*
- **Saves** - *model-specific functions*

f_build_model_exp

Code

```
function f_build_model_exp(stg,sb,mmf)
%Creates two .mat files for each experiment, with all the added rules,
%species and parameters needed depending on the inputs and outputs
%specified on the sbtab, one for the equilibrium simulation run and one for
%the proper run

data_model = mmf.model.data.data_model;
mat_model = mmf.model.data.mat_model;
model_exp_eq = mmf.model.data.model_exp.equilibration;
```

(continues on next page)

(continued from previous page)

```
model_exp_default = mmf.model.data.model_exp.default;
model_exp_detail = mmf.model.data.model_exp.detail;

%Find correct path for loading depending on the platform
load(data_model,'Data','sbtab')
load(mat_model,'modelobj');

model_run = cell(size(sb.Experiments.ID,1),1);
configsetObj = cell(size(sb.Experiments.ID,1),1);

for number_exp = 1:size(sb.Experiments.ID,1)

    output_value = sbtab.datasets(number_exp).output_value;
    output = sbtab.datasets(number_exp).output;
    input_time = sbtab.datasets(number_exp).input_time;
    input_value = sbtab.datasets(number_exp).input_value;
    input_species = sbtab.datasets(number_exp).input;

    model_run{number_exp} = copyobj(modelobj);
    configsetObj{number_exp} = getconfigset(model_run{number_exp});

    set(configsetObj{number_exp}, 'MaximumWallClock', stg.maxt);
    set(configsetObj{number_exp}, 'StopTime', stg.eqt);
    set(configsetObj{number_exp}.CompileOptions,...
        'DimensionalAnalysis', stg.dimenanal);
    set(configsetObj{number_exp}.CompileOptions,...
        'UnitConversion', stg.UnitConversion);
    set(configsetObj{number_exp}.SolverOptions,...
        'AbsoluteToleranceScaling', stg.abstolscale);
    set(configsetObj{number_exp}.SolverOptions,...
        'RelativeTolerance', stg.reltol);
    set(configsetObj{number_exp}.SolverOptions,...
        'AbsoluteTolerance', stg.abstol);
    set(configsetObj{number_exp}.SolverOptions, 'OutputTimes', stg.
↪eqt);
    set(configsetObj{number_exp}, 'TimeUnits', stg.simtime);
    set(configsetObj{number_exp}.SolverOptions,...
        'MaxStep', stg.maxstepeq);

    set(configsetObj{number_exp}.SolverOptions,...
        'AbsoluteToleranceStepSize', stg.abstolstepsize_eq);


    model_exp = model_run{number_exp};
    config_exp = configsetObj{number_exp};

    save(model_exp_eq + number_exp + ".mat",'model_exp','config_exp')

    sbiosaveproject(model_exp_eq + number_exp + ".sbproj",'model_exp')

    set(configsetObj{number_exp}, 'StopTime', sbtab.sim_time(number_
↪exp));
```

(continues on next page)

(continued from previous page)

```
60
61      set(configsetObj{number_exp}.SolverOptions, 'OutputTimes',...
62          Data(number_exp).Experiment.t);
63
64      set(configsetObj{number_exp}.SolverOptions, 'MaxStep', stg.
    ↪maxstep);
65
66      for n = 1:size(output,2)
67
68          m = 0;
69          for k = 1:size(model_run{number_exp}.species,1)
70              if model_run{number_exp}.species(k).name == string(output
    ↪{1,n})
71                  model_run{number_exp}.species(k).BoundaryCondition = 1;
72                  m = 1;
73              end
74          end
75
76          if m == 0
77              addspecies (model_run{number_exp}.Compartments(1),...
78                  char(output{1,n}),0,...
79                  'InitialAmountUnits',sb.Output.Unit{n});
80          end
81
82          addrule(model_run{number_exp}, char(output_value{1,n}),...
83              'repeatedAssignment');
84      end
85
86      for j = 1:size(input_species,2)
87
88          input_indexcode = str2double(strrep(input_species(j),'S',''));
89          input_name = string(model_run{number_exp}.species(1 +...
90                  input_indexcode).name);
91
92          if size(input_time{j},2) < 100
93
94              model_run{number_exp}.species(...
95                  1 + input_indexcode).BoundaryCondition = 1;
96              for n = 1:size(input_time{j},2)
97                  if ~isnan(input_time{j}(n))
98
99                      addparameter(model_run{number_exp},char("time_
    ↪event_t_" + j + "_" +  n),...
100                         str2double(string(input_time{j}(n))),...
101                         'ValueUnits',char(stg.simtime));
102
103                     addparameter(model_run{number_exp},char("time_
    ↪event_r_" + j + "_" +  n),...
104                         str2double(string(input_value{j}(n))),...
105                         'ValueUnits',char(model_run{number_exp}.
    ↪species(1 +...
106                         input_indexcode).InitialAmountUnits));
```

(continues on next page)

(continued from previous page)

```
                    addevent(model_run{number_exp}, ...
                        char("time>=time_event_t_" + j + "_" +  n),...
                        cellstr(sbtab.datasets(number_exp).output_
↪location{1} +...
                        "." + input_name + " = time_event_r_" + j + "_
↪" +  n));

                end
            end
        else

            addrule(model_run{number_exp}, char(sbtab.datasets(...
                number_exp).output_location{1} + "." + input_name +...
                "=" + string(model_run{number_exp}.name)+ "_input" + ..
↪.
                number_exp + "_" + input_name + "(time)"),
↪'repeatedAssignment');
        end
    end

    model_exp = model_run{number_exp};
    config_exp = configsetObj{number_exp};

    save(model_exp_default + number_exp + ".mat",'model_exp','config_
↪exp')

    sbiosaveproject(model_exp_default + number_exp + ".sbproj",'model_
↪exp')

    set(configsetObj{number_exp}.SolverOptions,'OutputTimes', []);
    set(configsetObj{number_exp}.SolverOptions,'MaxStep', stg.
↪maxstepdetail);

    model_exp = model_run{number_exp};
    config_exp = configsetObj{number_exp};

    save(model_exp_detail + number_exp + ".mat",'model_exp','config_exp
↪')

end
end
```

Creates two .mat files for each experiment, one for the *equilibrium simulation run* and one for the *proper simulation*. These files have all the added rules, species and parameters needed depending on the inputs and outputs specified on the SBtab.

- **Inputs** - *stg*, sb
- **Saves** - *Ready to run models*

Simulation and Scoring

f_sim_score

Code

```
function [score,rst,rst_not_simd] = f_sim_score(parameters,stg,mmf)

%Turn off Dimension analysis warning from simbiology
warning('off','SimBiology:DimAnalysisNotDone_MatlabFcn_Dimensionless')

% Call the function that simulates the model
rst = f_prep_sim(parameters,stg,mmf);

% Call the function that scores
rst = f_score(rst,stg,mmf);

% Get the total score explicitly for optimization functions
score = rst.st;

rst_not_simd = rmfield( rst , 'simd');
end
```

Calls the function that runs the simulations and the function that scores the output of the runs.

- **Inputs**
  - *stg*
  - parameters - (double) Set of parameters that we are working on
- **Outputs**
  - score - *rst.st*
  - *rst* - *rst.simd*, *rst.xfinal*, *rst.sd*, *rst.se*, and *rst.st*
  - rst_not_simd - *rst.xfinal*, *rst.sd*, *rst.se*, and *rst.st*
- **Calls** - *f_prep_sim*, *f_score*
- **Loads**

f_prep_sim

Code

```
function rst = f_prep_sim(parameters,stg,mmf)

% Save variables that need to be mantained over multiple function calls
% persistent modelobj
persistent sbtab
persistent Data

data_model = mmf.model.data.data_model;

% Import the data on the first run
```

(continues on next page)

(continued from previous page)

```
if isempty(sbtab)

    %Find correct path for loading depending on the platform
    load(data_model,'Data','sbtab')
end

% Set the parameters that are going to be used for the simulation to␣
↪the
% default ones as definded in the SBTAB
rt.par(:,1) = [sbtab.defpar{:,2}];

% Check if the parametrer needs to be set to the value relevant for␣
↪Profile
% Likelihood
if isfield(stg,"PLind") && isfield(stg,"PLval")
    parameters = [parameters(1:stg.PLind-1) stg.PLval parameters(stg.
↪PLind:end)];
end

% Iterate over all the parameters of the model
for n = 1:size(rt.par,1)

    % Check that a parameter should be changed from default
    if stg.partest(n) > 0

        % Set the parameters are being tested
        rt.par(n) = 10.^(parameters(stg.partest(n,1)));
    end

    if isfield(stg,'tci')

        % Check that there are thermodynamic constraints to implement
        if ~isempty(stg.tci)

            % Choose the parameters that need to be calculated with␣
↪other
            % parameters due to thermodynamic constraints
            if ismember(n,stg.tci)

                % Check that a parameter should be changed from default
                if stg.partest(n) > 0

                    % Iterate over the parameters that need to be␣
↪mutiplied
                    % for calculating the parameter that depends on the
                    % thermodynamic constraints
                    for m = 1:size(stg.tcm,2)

                        % Check that the parameter that is going to be␣
↪used
                        % to calculate the parameter dependent on
                        % thermodynamic constraintsis is not the␣
↪default
```

(continues on next page)

(continued from previous page)

```
                        if stg.partest(stg.tcm(n,m),1) > 0

                            % Check if the parametrer needs to be set␣
↪to
                            % the value relevant for Profile Likelihood
                            if isfield(stg,"PLind")
                                if stg.partest(stg.tcm(n,m),1) ==...
                                        stg.PLind
                                    parameters(stg.partest(...
                                        stg.tcm(n,m),1))...
                                        = stg.PLval;
                                end
                            end

                            % Make the appropriate multiplications to␣
↪get
                            % the thermodinamicly constrained parameter
                            rt.par(n) = rt.par(n).*(10.^...
                                (parameters(stg.partest(...
                                stg.tcm(n,m),1))));
                        else

                            % Make the appropriate multiplications to␣
↪get
                            % the thermodinamicly constrained parameter
                            rt.par(n) = rt.par(n).*...
                                (sbtab.defpar{stg.tcm(n,m),2});
                        end
                    end

                    % Iterate over the parameters that need to be␣
↪divided
                    % for calculating the parameter that depends on the
                    % thermodynamic constraints
                    for m = 1:size(stg.tcd,2)

                        % Check that the parameter that is going to be␣
↪used
                        % to calculate the parameter dependent on
                        % thermodynamic constraintsis is not the␣
↪default
                        if stg.partest(stg.tcd(n,m),1) > 0

                            % Check if the parametrer needs to be set␣
↪to
                            % the value relevant for Profile Likelihood
                            if isfield(stg,"PLind")
                                if stg.partest(stg.tcd(n,m),1) ==...
                                        stg.PLind
                                    parameters(stg.partest(...
                                        stg.tcd(n,m),1))...
                                        = stg.PLval;
```

(continues on next page)

(continued from previous page)

```
                                end
                            end
                            % Make the appropriate divisions to get the
                            % thermodinamicly constrained parameter
                            rt.par(n) = rt.par(n)./(10.^...
                                (parameters(stg.partest(...
                                stg.tcd(n,m),1))));
                        else

                            % Make the appropriate divisions to get the
                            % thermodinamicly constrained parameter
                            rt.par(n) = rt.par(n)./...
                                (sbtab.defpar{stg.tcd(n,m),2});
                        end
                    end
                end
            end
        end
    end
end

%  Set the start amount for the species in the model to 0
rt.ssa = zeros(size(sbtab.species,1),max(stg.exprun));

% Initialize the results variable
rst = [];
rst.parameters = rt.par;
% Iterate over all the experiments that are being run
for n = stg.exprun

    % Try catch used because iterations errors can happen unexectedly␣
↪and
    % we want to be able to continue simulations
    try

        % If the correct setting is chosen display messages to console
        if stg.simcsl
            disp("Running dataset number " + n +...
                " of " + stg.exprun(end))
        end

        % Check that this is not the first time the experiments are␣
↪being
        % run and that the start values for the species are different␣
↪from
        % the previous experiment
        if n ~= stg.exprun(1) && ...
                min([sbtab.datasets(n).start_amount{:,2}] ==...
                [sbtab.datasets(max(1,stg.exprun(...
                find(stg.exprun==n)-1))).start_amount{:,2}])


```

(continues on next page)

(continued from previous page)

```
            % Set the values of the start amounts to the values␣
↪obtained
            % after the first equilibration
            rt.ssa(:,n) =...
                rt.ssa(:,stg.exprun(find(stg.exprun==n)-1));
            if stg.simdetail
                rt.ssa(:,n+2*stg.expn) =...
                    rt.ssa(:,stg.exprun(find(stg.exprun==n)-1));
            end
        else

            % Iterate over the numbre of species that need a starting␣
↪value
            % different than 0
            for j = 1:size(sbtab.datasets(n).start_amount,1)

                % Set the start amount of the species to the number␣
↪defined
                % in the sbtab for each experiment
                rt.ssa(sbtab.datasets(n...
                    ).start_amount{j,3},n+stg.expn) =...
                    sbtab.datasets(n).start_amount{j,2};
            end

            % Equilibrate the model
            rst = f_sim(n+stg.expn,stg,rt,rst,mmf);

            for j = 1:size(sbtab.species,1)

                % Set the starting amount for species that after
                % equilibrium have very low values to zero
                if rst.simd{n+stg.expn}.Data(end,j) < 1.0e-15
                    rt.ssa(j,n) = 0;

                    % Set the starting amount for the rest of the␣
↪species
                else
                    rt.ssa(j,n) =...
                        rst.simd{n+stg.expn}.Data(end,j);
                    if stg.simdetail
                        rt.ssa(j,n+2*stg.expn) =...
                            rst.simd{n+stg.expn}.Data(end,j);
                    end
                end
            end
        end

        % Simulate the model
        rst = f_sim(n,stg,rt,rst,mmf);

        try
            if stg.simdetail
```

(continues on next page)

(continued from previous page)

```
                rst = f_sim(n+2*stg.expn,stg,rt,rst,mmf);
            end
        catch
        end

        % Check If the times of the simultaion output and the␣
↪simulation
        % data from SBTAB match, if they don't it means that the␣
↪simulator
        % didn't had enough time to run the model (happens in some
        % unfavorable configurations of parameters, controlled by stg.
↪maxt
        if size(Data(n).Experiment.t,1) ~=...
                size(rst.simd{n}.Data(:,end),1)

            % Set the simulation output to be 0, this is a non function
            % value that the score function expects in simulations␣
↪that did
            % not worked properly
            rst.simd{n} = 0;
        end
    catch

        % Set the simulation output to be 0, this is a non function␣
↪value
        % that the score function expects in simulations that did not
        % worked properly
        rst.simd{n} = 0;
    end
end
end
```

Prepares the simulation making sure that an equilibration is preformed when necessary before running the main simulation.

- **Inputs**
  - *stg* - stg.folder_model, *stg.name*, *stg.partest*, *stg.tci*, *stg.tcm*, *stg.tcd*, *stg.exprun*, *stg.simcsl*, *stg.expn*
  - parameters - (double) Set of parameters that we are working on
- **Created Variables**
  - rt
    - rt.ssa - (double) steady state amounts
    - rt.par - (double) All parameters of the model, takes the default ones from SBtab and then replaces the ones being worked on.
- **Outputs**
  - *rst* - *rst.simd*
- **Calls** - *f_sim*
- **Loads** - *data.mat*, *model.mat*

f_sim

Code

```
function rst = f_sim(exp_n,stg,rt,rst,mmf)

% Save variables that need to be mantained over multiple function calls
persistent model_run
persistent config_run

% If the function is called for the first time, load the appropriate model
% and compile the code for simulation run
if isempty(model_run)

    %Generate an empty array to be populated with the model suited for each
    %equilibration and experiment%
    model_run = cell(1,stg.expn*2);
    config_run = cell(1,stg.expn*2);

    model_exp_default = mmf.model.data.model_exp.default;
    model_exp_eq = mmf.model.data.model_exp.equilibration;
    model_exp_detail = mmf.model.data.model_exp.detail;

    % Iterate over the experiments thar are being run
    for n = stg.exprun

        if stg.simdetail
            % Load the models for equilibrium
            load(model_exp_detail + n + ".mat",'model_exp','config_exp')

            % Place the loaded models in the correct place in the array,
            % models for equilibrium are set to be on the second half of
            % the array
            model_run{n+2*stg.expn} = model_exp;
            config_run{n+2*stg.expn} = config_exp;

            % Compile the matlab code that is going to simulate the model
            % using matlab built in function if the option is selected in
            % settings
            if stg.sbioacc
                sbioaccelerate(model_run{n+2*stg.expn},config_run{n+2*stg.expn});
            end
        end
        % Load the models for equilibrium
        load(model_exp_eq + n + ".mat",'model_exp','config_exp')
```

(continues on next page)

(continued from previous page)

```
42
43          % Place the loaded models in the correct place in the array, models
44          % for equilibrium are set to be on the second half of the array
45          model_run{n+stg.expn} = model_exp;
46          config_run{n+stg.expn} = config_exp;
47
48          % Compile the matlab code that is going to simulate the model using
49          % matlab built in function if the option is selected in settings
50          if stg.sbioacc
51              sbioaccelerate(model_run{n+stg.expn},config_run{n+stg.expn});
52          end
53
54          % Load the models for main run
55          load(model_exp_default + n + ".mat",'model_exp','config_exp')
56
57          % Place the loaded models in the correct place in the array, models
58          % for main run are set to be on the first half of the array
59          model_run{n} = model_exp;
60          config_run{n} = config_exp;
61
62          % Compile the matlab code that is going to simulate the model using
63          % matlab built in function if the option is selected in settings
64          if stg.sbioacc
65              sbioaccelerate(model_run{n},config_run{n});
66          end
67      end
68  end
69
70  % substitute the start amount of the species in the model with the correct
71  % ones for  simulations
72  set(model_run{exp_n}.species(1:size(rt.ssa(:,exp_n),1)),{'InitialAmount'},num2cell(rt.ssa(:,exp_n)))
73
74  % Substitute the values of the parameters in the model for the correct one
75  % for simultaions
76  set(model_run{exp_n}.parameters(1:size(rt.par,1)),{'Value'},num2cell(rt.par))
77
78  %simulate the model using matlab built in function
79  rst.simd{exp_n} = sbiosimulate(model_run{exp_n},config_run{exp_n});
80  end
```

Simulates the model with the provided configurations. The first time it is run it loads a representation of the model and the simulation, and compiles this information to C code.

- **Inputs**
  - exp_n - (double) Unique number to identify the model for each experiment or equilibrium reaction (it needs a new model object for each one)
  - *stg* - *stg.expn*, stg.folder_model, *stg.name*, *stg.sbioacc*
  - rt
    - rt.ssa - (double) steady state amounts
    - rt.par - (double) All parameters of the model, takes the default ones from SBtab and then replaces the ones being worked on.
  - *rst* - *rst.simd*
- **Outputs**
  - *rst* - *rst.simd*
- **Calls** - Sbioaccelerate, Sbiosimulate
- **Loads** - *Ready to run model*, *Ready to run model equilibration*

f_score

Code

```
function rst = f_score(rst,stg,mmf)

persistent sbtab
persistent Data

data_model = mmf.model.data.data_model;

% Import the data on the first run
if isempty(Data)
    load(data_model,'Data','sbtab')
end

% Iterate over the number of experiments
for n = stg.exprun

    % Iterate over the number of datasets per experiment
    for j = 1:size(sbtab.datasets(n).output,2)

        % Calculate score per dataset if there are no errors
        if rst.simd{n} ~= 0

            data = Data(n).Experiment.x(:,j);
            data_sd = Data(n).Experiment.x_SD(:,j);
            number_points = size(Data(n).Experiment.x(:,j),1);
            sim_results = f_normalize(rst,stg,n,j,mmf);
            rst.xfinal{n,1}(j) = sim_results(end);

            % Calculate score using formula that accounts for␣
↪normalization
```

(continues on next page)

(continued from previous page)

```
            % with the starting point of the result
            if stg.useLog == 4
                rst.sd{n,1}(j) = sum(((data-sim_results)./...
                    (data_sd*sqrt(number_points))).^2);
            else
                    rst.sd{n,1}(j) = sum(((data-sim_results)./...
                        (data_sd)).^2)/(number_points);
            end

            % If there are errors output a very high score value (10^
↪10)
        elseif rst.simd{n} == 0 || rst.sd{n,1}(j) == inf

                rst.sd{n,1}(j) = stg.errorscore;
                rst.xfinal{n,1}(j) = 0;
        end

        % Calculate the log10 of dataset score if option selected
        if stg.useLog == 1
            rst.sd{n,1}(j) = max(0,log10(rst.sd{n,1}(j)));
        end
    end

    % Calculate score per experiment
    rst.se(n,1) = sum(rst.sd{n,1});

    % Calculate the log10 of experiment score if option selected
    if stg.useLog == 2
        rst.se(n,1) = log10(rst.se(n,1));
    end
end

% Calculate score per experiment
rst.st = sum(rst.se);

% Calculate the log10 of total score if option selected
if stg.useLog == 3
    rst.st = log10(rst.st);
end
end
```

Uses the results from the simulation of the model and the Data provided via the SBTAB to calculate a score for a given parameter set.

- **Inputs**
  - *rst* - *rst.simd*
  - *stg* - stg.folder_model, *stg.name*, *stg.exprun*, *stg.useLog*
- **Outputs**
  - *rst.st* - *rst.xfinal*, *rst.sd*, *rst.se*, *rst.st*
- **Calls**

- **Loads** - *data.mat*

Analysis

f_analysis

Code

```
function rst = f_analysis(stg,analysis,mmf,analysis_options)

if contains(analysis,analysis_options(1))
    disp("Starting " + analysis_options(1))
    rst.diag = f_diagnostics(stg,mmf);
    disp(analysis_options(1) + " completed successfully")
end

if contains(analysis,analysis_options(2))
    disp("Starting " + analysis_options(2))
    rst.opt = f_opt(stg,mmf);
    disp(analysis_options(2) + " completed successfully")
end

if contains(analysis,analysis_options(3))
    disp("Starting " + analysis_options(3))
    rst.gsa = f_gsa(stg,mmf);
    disp(analysis_options(3) + " completed successfully")
end
end
```

Calls the proper analysis functions depending on the analysis that was chosen on the settings file. The supported analysis right now are:

- *Model diagnostics functions*
- *Optimization*
- *Global sensitivity analysis*
- **Inputs**
  - *stg*
  - analysis - (string) analysis being run (*stg.analysis*)
- **Outputs** - *rst*

Diagnostics

f_diagnostics

Code

```
function rst = f_diagnostics(stg,mmf)

% Run the model and obtain scores for fitness Multi Core
```

(continues on next page)

(continued from previous page)

```
if stg.optmc
    disp("Running the model and obtaining Scores (Multicore)")

    pa = stg.pa;
    % Iterate over the parameter arrays to be tested
    parfor n = stg.pat
        [~,rst(n),~] = f_sim_score(pa(n,:),stg,mmf);
    end

    % Run the model and obtain scores for fitness single Core
else
    disp("Running the model and obtaining Scores (Singlecore)")

    % Iterate over the parameter arrays to be tested
    for n = stg.pat
        [~,rst(n),~] = f_sim_score(stg.pa(n,:),stg,mmf);
    end
end
end
```

Used to understand the effects of different parameters sets on model behaviour or in comparing different parameters sets.
It loads the user defined configurations, performs all the needed simulations, and calculates scores of the error functions either per experimental output, per experiment, or in total (*check results*).

- **Inputs**
  - *stg* - *stg.optmc* , *stg.pat*
- **Outputs** - rst (*diagnostics results*)

Optimization

f_opt

Code

```
function rst = f_opt(stg,mmf)
% Call function to run fmincon optimization algorithm if chosen in␣
↪settings

if stg.fmincon
    rst(1) = f_opt_fmincon(stg,mmf);
end

% Call function to run simulated annealing optimization algorithm if␣
↪chosen
% in settings
if stg.sa
    rst(2) = f_opt_sa(stg,mmf);
```

(continues on next page)

(continued from previous page)

```
end

% Call function to run pattern search optimization algorithm if chosen in
% settings
if stg.psearch
    rst(3) = f_opt_psearch(stg,mmf);
end

% Call function to run genetic algorithm optimization if chosen in settings
if stg.ga
    rst(4) = f_opt_ga(stg,mmf);
end

% Call function to run Particle swarm optimization algorithm if chosen in
% settings
if stg.pswarm
    rst(5) = f_opt_pswarm(stg,mmf);
end

% Call function to run Surrogate optimization algorithm if chosen in
% settings
if stg.sopt
    rst(6) = f_opt_sopt(stg,mmf);
end
end
```

Calls the correct optmizer or optimizers that have been chosen in the settings file.

- **Inputs**
  - *stg* - *stg.fmincon*, *stg.sa*, *stg.psearch*, *stg.ga*, *stg.pswarm*, *stg.sopt*
- **Outputs** - rst (*optimization results*)

f_opt_start

Code

```
function [spoint,spop] = f_opt_start(stg)

% Set the randomm seed for reproducibility
rng(stg.rseed);

% Optimization Start method 1
if stg.osm == 1

    % Get a random starting point or group of starting points, if using
    % multistart, inside the bounds
    spoint = lhsdesign(stg.msts,stg.parnum).*(stg.ub-stg.lb)+stg.lb;

```

(continues on next page)

(continued from previous page)

```
    % Get a group of ramdom starting points inside the bounds
    spop = lhsdesign(stg.popsize,stg.parnum).*(stg.ub-stg.lb)+stg.lb;

    % Optimization Start method 2
elseif stg.osm == 2

    % Get a random starting point or group of starting points, if using
    % multistart, near the best point
    spoint = stg.bestpa - stg.dbpa +...
        (stg.dbpa*2*lhsdesign(stg.msts,stg.parnum));

    % Get a group of ramdom starting points near the best point
    spop = stg.bestpa - stg.dbpa +...
        (stg.dbpa*2*lhsdesign(stg.popsize,stg.parnum));
end
end
```

Creates the starting parameter set or sets of the optimizations, if single or multistart selected in settings file. It supports two different random distributions for the starting points.

- **Inputs**
  - *stg* - *stg.rseed*, *stg.osm*, *stg.msts*, *stg.parnum*, *stg.ub*, *stg.lb*, *stg.popsize*, *stg.bestpa*, *stg.dbpa*
- **Outputs**
  - spoint - (double) starting parameter set for the optimization
  - spop - (double) Starting parameter sets for multiple start optimizations

f_opt_fmincon/sa/psearch/ga/pswarm/sopt

Code

f_opt_fmincon

```
function rst = f_opt_fmincon(stg,mmf)

% Set the randomm seed for reproducibility
rng(stg.rseed);

% Set the starting point for the optimization
[startpoint,~] = f_opt_start(stg);

% Get the optimization options from settings
options = stg.fm_options;
options.UseParallel = stg.optmc;

% Display console messages if chosen in settings
if stg.optcsl
    options.Display = 'iter-detailed';
```

(continues on next page)

(continued from previous page)

```
end

% Display plots if chosen in settings
if stg.optplots
    options.PlotFcn = ...
        {@optimplotx,@optimplotfunccount,@optimplotfval,...
        @optimplotstepsize,@optimplotfirstorderopt};
end

% Optimize the model with multiple starting points if chosen in␣
↪settings
if stg.mst
    parfor n = 1:stg.msts
        disp(string(n))
        [x(n,:),fval(n),exitflag(n),output(n)] =...
            fmincon(@(x)f_sim_score(x,stg,mmf),startpoint(n,:),...
            [],[],[],[],stg.lb,stg.ub,[],options);
    end

    % Optimize the model
else
    [x(1,:),fval(1),exitflag(1),output(1)] =...
        fmincon(@(x)f_sim_score(x,stg,mmf),startpoint(1,:),...
        [],[],[],[],stg.lb,stg.ub,[],options);
end

% Save results
rst.name = 'fmincon';
rst.x = x;
rst.fval = fval;
rst.exitflag = exitflag;
rst.output = output;
end
```

f_opt_sa

```
function rst = f_opt_sa(stg,mmf)

% Set the randomm seed for reproducibility
rng(stg.rseed);

% Set the starting point for the optimization
[startpoint,~] = f_opt_start(stg);

% Get the optimization options from settings
options = stg.sa_options;
options.MaxTime = stg.optt;
options.InitialTemperature = ones(1,stg.parnum)*2;

% Display console messages if chosen in settings
if stg.optcsl
    options.Display = 'iter';
```

(continues on next page)

(continued from previous page)

```
end

% Display plots if chosen in settings
if stg.optplots
    options.PlotFcn =...
        {@saplotbestf,@saplottemperature,@saplotf,@saplotstopping,
↪@saplotx};
end

% Optimize the model with multiple starting points if chosen in␣
↪settings
if stg.mst
    parfor n = 1:stg.msts
        disp(string(n))
        [x(n,:),fval(n),exitflag(n),output(n)] =...
            simulannealbnd(@(x)f_sim_score(x,stg,mmf),startpoint(n,:),.
↪..
            stg.lb,stg.ub,options);
    end

    % Optimize the model
else
    [x(1,:),fval(1),exitflag(1),output(1)] =...
        simulannealbnd(@(x)f_sim_score(x,stg,mmf),startpoint(1,:),...
        stg.lb,stg.ub,options);
end

% Save results
rst.name = 'Simulated annealing';
rst.x = x;
rst.fval = fval;
rst.exitflag = exitflag;
rst.output = output;
end
```

f_opt_psearch

```
function rst = f_opt_psearch(stg,mmf)

% Set the randomm seed for reproducibility
rng(stg.rseed);

% Set the starting point for the optimization
[startpoint,~] = f_opt_start(stg);

% Get the optimization options from settings
options = stg.psearch_options;
options.MaxTime = stg.optt;
options.UseParallel = stg.optmc;

% Display console messages if chosen in settings
if stg.optcsl
```

(continues on next page)

(continued from previous page)

```
    options.Display = 'iter';
end

% Display plots if chosen in settings
if stg.optplots
    options.PlotFcn = ...
        {@psplotbestf,@psplotfuncount,@psplotmeshsize,@psplotbestx};
end

% Optimize the model with multiple starting points if chosen in settings
if stg.mst
    parfor n = 1:stg.msts
        disp(string(n))
        [x(n,:),fval(n),exitflag(n),output(n)] =...
            patternsearch(@(x)f_sim_score(x,stg,mmf),startpoint(n,:),[],[], ...
            [],[],stg.lb,stg.ub,[],options);
    end

    % Optimize the model
else
    [x(1,:),fval(1),exitflag(1),output(1)] =...
        patternsearch(@(x)f_sim_score(x,stg,mmf),startpoint(1,:),[],[], ...
        [],[],stg.lb,stg.ub,[],options);
end

% Save results
rst.name = 'Pattern search';
rst.x = x;
rst.fval = fval;
rst.exitflag = exitflag;
rst.output = output;
end
```

f_opt_ga

```
function rst = f_opt_ga(stg,mmf)

% Set the randomm seed for reproducibility
rng(stg.rseed);

% Get the optimization options from settings
options = stg.ga_options;
options.MaxTime = stg.optt;
options.UseParallel = stg.optmc;
options.PopulationSize = stg.popsize;

% Display console messages if chosen in settings
if stg.optcsl
    options.Display = 'iter';
```

(continues on next page)

(continued from previous page)

```
end

% Display plots if chosen in settings
if stg.optplots
    options.PlotFcn = {@gaplotbestf,...
        @gaplotexpectation,@gaplotrange,@gaplotscorediversity,...
        @gaplotstopping,@gaplotscores,@gaplotdistance,...
        @gaplotselection};
end

% Set the starting population for the optimization
[~,startpop] = f_opt_start(stg);
options.InitialPopulationMatrix = startpop;

% Optimize the model with multiple starting points if chosen in␣
↪settings
if stg.mst
    parfor n = 1:stg.msts
        disp(string(n))
        [x(n,:),fval(n)] = ga(@(x)f_sim_score(x,stg,mmf),stg.parnum,...
            [],[],[],[],stg.lb,stg.ub,[],options);
    end

    % Optimize the model
else
    [x(1,:),fval(1)] = ga(@(x)f_sim_score(x,stg,mmf),stg.parnum,...
        [],[],[],[],stg.lb,stg.ub,[],options);
end

% Save results
rst.name = 'Genetic algorithm';
rst.x = x;
rst.fval = fval;
rst.exitflag = [];
rst.output = [];
end
```

f_opt_pswarm

```
function rst = f_opt_pswarm(stg,mmf)

% Set the randomm seed for reproducibility
rng(stg.rseed);

% Get the optimization options from settings
options = stg.pswarm_options;
options.MaxTime = stg.optt;
options.UseParallel = stg.optmc;
options.SwarmSize = stg.popsize;

% Display console messages if chosen in settings
if stg.optcsl
```

(continues on next page)

(continued from previous page)

```
    options.Display = 'iter';
end

% Display plots if chosen in settings
if stg.optplots
    options.PlotFcn = @pswplotbestf;
end

% Set the starting population for the optimization
[~,startpop] = f_opt_start(stg);
options.InitialSwarmMatrix = startpop;

% Optimize the model with multiple starting points if chosen in␣
↪settings
if stg.mst
    parfor n = 1:stg.msts
        disp(string(n))
        [x(n,:),fval(n),exitflag(n),output(n)] =...
            particleswarm(@(x)f_sim_score(x,stg,mmf),...
            stg.parnum,stg.lb,stg.ub,options);
    end

    % Optimize the model
else
    [x(1,:),fval(1),exitflag(1),output(1)] =...
        particleswarm(@(x)f_sim_score(x,stg,mmf),...
        stg.parnum,stg.lb,stg.ub,options);
end

% Save results
rst.name = 'Particle swarm';
rst.x = x;
rst.fval = fval;
rst.exitflag = exitflag;
rst.output = output;
end
```

f_opt_sopt

```
function rst = f_opt_sopt(stg,mmf)

% Set the randomm seed for reproducibility
rng(stg.rseed);

% Get the optimization options from settings
options = stg.sopt_options;
options.MaxTime = stg.optt;
options.UseParallel = stg.optmc;

% Display console messages if chosen in settings
if stg.optcsl
    options.Display = 'iter';
```

(continues on next page)

(continued from previous page)

```
end

% Display plots if chosen in settings
if stg.optplots
    options.PlotFcn = @surrogateoptplot;
end

% Set the starting population for the optimization
[~,startpop] = f_opt_start(stg);
options.InitialPoints = startpop;

% Optimize the model with multiple starting points if chosen in␣
↪settings
if stg.mst
    parfor n = 1:stg.msts
        disp(string(n))
        [x(n,:),fval(n),exitflag(n),output(n)] =...
            surrogateopt(@(x)f_sim_score(x,stg,mmf),stg.lb,stg.ub,
↪options);
    end

    % Optimize the model
else
    [x(1,:),fval(1),exitflag(1),output(1)] =...
        surrogateopt(@(x)f_sim_score(x,stg,mmf),stg.lb,stg.ub,options);
end

% Save results
rst.name = 'Surrogate optimization';
rst.x = x;
rst.fval = fval;
rst.exitflag = exitflag;
rst.output = output;
end
```

These functions call built in MATLAB® functions that perform parameter optimization . For furher information relating to how these optimizers work please follow the links to the MATLAB® documentation. Optimizers used:

- f_opt_fmincon - fmincon
- f_opt_sa - Simmulated annealing
- f_opt_psearch - Pattern search
- f_opt_ga - Genetic algorihtm
- f_opt_pswarm - Particle swarm
- f_opt_sopt - Surrogate optmization
- **Inputs** - *stg*
- **Outputs** - *Optimization results*

Global Sensitivity Analysis

f_gsa

Code

```
function rst = f_gsa(stg,mmf)

rst = f_make_par_samples(stg);

rst = f_make_output_sample(rst,stg,mmf);

rst = f_calc_sensitivities(rst,stg);
end
```

Calls the global sensitivity analysis functions in the correct order.

f_make_par_samples

Code

```
function rst= f_make_par_samples(stg)
% Code inspired by Geir Halnes et al. 2009 paper. (Halnes, Geir, et al.
↪ J.
% comp. neuroscience 27.3 (2009): 471.)

% MAKE SAMPLE MATRICES
M1 = zeros(stg.sansamples, stg.parnum); % Pre-allocate memory for data
M2 = zeros(stg.sansamples, stg.parnum);
N = zeros(stg.sansamples, stg.parnum, stg.parnum);
rng(stg.rseed)

% Create a distribution for each parameter acording to settings(Note␣
↪that
% the sampling is being performed in log space as the parameters and␣
↪its
% bounds are in log space)
for i=1:stg.parnum
    % Uniform distribution truncated at the parameter bounds
    if stg.sasamplemode == 0
        M1(:,i) = stg.lb(i) +...
            (stg.ub(i)-stg.lb(i)).*rand(1,stg.sansamples);
        M2(:,i) = stg.lb(i) +...
            (stg.ub(i)-stg.lb(i)).*rand(1,stg.sansamples);
        % Normal distribution with mu as the best value for a␣
↪parameter and
        % sigma as stg.sasamplesigma truncated at the parameter bounds
    elseif stg.sasamplemode == 1
        pd(i) = makedist('Normal','mu',stg.bestpa(i),...
            'sigma',stg.sasamplesigma);
        t(i) = truncate(pd(i),stg.lb(i),stg.ub(i));
        r{i} = random(t(i),stg.sansamples,1);
```

(continues on next page)

(continued from previous page)

```
        r2{i} = random(t(i),stg.sansamples,1);
        M1(:,i) = r{i};
        M2(:,i) = r2{i};
        % Same as 1 without truncation
    elseif stg.sasamplemode == 2
        pd(i) = makedist('Normal','mu',stg.bestpa(i),...
            'sigma',stg.sasamplesigma);
        r{i} = random(pd(i),stg.sansamples,1);
        r2{i} = random(pd(i),stg.sansamples,1);
        M1(:,i) = r{i};
        M2(:,i) = r2{i};
        % Normal distribution centered at the mean of the parameter␣
↪bounds
        % and sigma as stg.sasamplesigma truncated at the parameter␣
↪bounds
    elseif stg.sasamplemode == 3
        pd(i) = makedist('Normal','mu',...
            stg.lb(i) + (stg.ub(i)-stg.lb(i))/2,'sigma',stg.
↪sasamplesigma);
        t(i) = truncate(pd(i),stg.lb(i),stg.ub(i));
        r{i} = random(t(i),stg.sansamples,1);
        r2{i} = random(t(i),stg.sansamples,1);
        M1(:,i) = r{i};
        M2(:,i) = r2{i};
        % Same as 3 without truncation.
    elseif stg.sasamplemode == 4
        pd(i) = makedist('Normal','mu',...
            stg.lb(i) + (stg.ub(i)-stg.lb(i))/2,'sigma',stg.
↪sasamplesigma);
        r{i} = random(pd(i),stg.sansamples,1);
        r2{i} = random(pd(i),stg.sansamples,1);
        M1(:,i) = r{i};
        M2(:,i) = r2{i};
    end
end

for i=1:stg.parnum
    % Replace the i:th column in M2 by the i:th column from M1 to␣
↪obtain Ni
    N(:,:,i) = M2;
    N(:,i,i) = M1(:,i);
end

rst.M1=M1;
rst.M2=M2;
rst.N=N;
```

Creates parameter sets samples with *specific parameter distributions* that are used to perform the global sensitivity analysis.

- **Inputs**
  - stg - *stg.sansamples*, *stg.parnum*, *stg.sasamplemode*, *stg.ub*, *stg.lb*

- **Outputs** - *M1*, *M2*, *N*

Code inspired by Geir Halnes et al. 2009 paper.

f_make_output_sample

Code

```
function rst = f_make_output_sample(rst,stg,mmf)
% Code inspired by Geir Halnes et al. 2009 paper. (Halnes, Geir, et al.
↪ J.
% comp. neuroscience 27.3 (2009): 471.)

nSamples = stg.sansamples;
nPars = stg.parnum;
parameter_array = zeros(nSamples,nPars);
progress = 1;
time_begin = datetime;
D = parallel.pool.DataQueue;

afterEach(D, @progress_track);

for i=1:nSamples
    parameter_array(i,:) = rst.M1(i,:);
end

parfor i=1:nSamples
    [~,~,RM1(i)] = f_sim_score(parameter_array(i,:),stg,mmf);
    send(D, "GSA M1 ");
end
disp("GSA M1 Runtime: " + string(datetime - time_begin) +...
                "  All " + nSamples + " samples executed")


for i=1:nSamples
    fM1.sd(i,:) = [RM1(i).sd{:}];
    fM1.se(i,:) = RM1(i).se(:);
    fM1.st(i,:) = RM1(i).st;
    fM1.xfinal(i,:) = [RM1(i).xfinal{:}];
end

rst.fM1 = fM1;
clear a FM1

for  i=1:nSamples
    parameter_array(i,:)= rst.M2(i,:);
end

progress = 1;
parfor i=1:nSamples
    [~,~,RM2(i)] = f_sim_score(parameter_array(i,:),stg,mmf);
    send(D, "GSA M2 ");
end
```

(continues on next page)

(continued from previous page)

```
disp("GSA M2 Runtime: " + string(datetime - time_begin) +...
                "  All " + nSamples + " samples executed")

for i=1:nSamples
    fM2.sd(i,:) = [RM2(i).sd{:}];
    fM2.se(i,:) = RM2(i).se(:);
    fM2.st(i,:) = RM2(i).st;
    fM2.xfinal(i,:) = [RM2(i).xfinal{:}];
end

rst.fM2 = fM2;
clear b FM2

for i=1:nSamples
    for j=1:nPars
        parameter_array(i,:,j)= rst.N(i,:,j);
    end
end

progress = 1;
parfor i=1:nSamples
    for j=1:nPars
        [~,~,RN{i,j}] = f_sim_score(parameter_array(i,:,j),stg,mmf);
    end
    send(D, "GSA N  ");
end
disp("GSA N  Runtime: " + string(datetime - time_begin) +...
                "  All " + nSamples + " samples executed")

for i=1:nSamples
    for j=1:nPars
        fN.sd(i,:,j) = [RN{i,j}.sd{:}];
        fN.se(i,:,j) = RN{i,j}.se(:);
        fN.st(i,:,j) = RN{i,j}.st;
        fN.xfinal(i,:,j) = [RN{i,j}.xfinal{:}];
    end
end

rst.fN = fN;
clear c FN

    function progress_track(name)
        progress = progress + 1;
        if mod(progress,ceil(nSamples/10)) == 0 && progress ~= nSamples
            disp(name + "Runtime: " + string(datetime - time_begin) +..
↪.
                "  Samples:" + progress + "/" + nSamples)
        end
    end
end
```

For each parameter set given in the matrices *M1*, *M2*, and *N* it runs the function *f_sim_score* generating new matrices *fM1*, *fM2*, and *fN* respectively.

- **Inputs** - *M1*, *M2*, *N*, *stg.sansamples*, *stg.parnum*,
- **Outputs** - *fM1*, *fM2*, *fN*

Code inspired by Geir Halnes et al. 2009 paper.

f_calc_sensitivities

Code

```
function rst = f_calc_sensitivities(rst,stg)

rst = remove_sim_error(rst,stg);

[rst.SiQ,rst.SiTQ,rst.Si,rst.SiT] = bootstrap(rst,stg);
end

function [SiQ,SiTQ,Si,SiT]=bootstrap(rst,stg)
% calculates confidence intervals.
fM1 = rst.fM1;
fM2 = rst.fM2;
fN = rst.fN;

scores_names_list = ["sd","se","st","xfinal"];

[Si,SiT] = bootstrap_h(fM1,fM2,fN,stg,scores_names_list);

if (isempty(stg.gsabootstrapsize))
    stg.gsabootstrapsize=ceil(sqrt(size(fM1.sd)));
end%if

fM1q = cell(stg.gsabootstrapsize,1);
fM2q = cell(stg.gsabootstrapsize,1);
fNq = cell(stg.gsabootstrapsize,1);

parfor j=1:stg.gsabootstrapsize
    [SiQh{j},SiTQh{j}] = bootstrap_hq(fM1,fM2,fN,stg,scores_names_list,
↪j);
end%parfor

for n = 1:size(scores_names_list,2)
    for j=1:stg.gsabootstrapsize
        eval("SiQ." + scores_names_list(n) + "(j,:,:) = SiQh{j}." +␣
↪scores_names_list(n) + ";")
        eval("SiTQ." + scores_names_list(n) + "(j,:,:) = SiTQh{j}." +␣
↪scores_names_list(n) + ";")
    end
end
end%function

function [Si,SiT] = bootstrap_h(fM1,fM2,fN,stg,scores_names)
for n = 1:size(scores_names,2)
    eval("fM1h=fM1." + scores_names(n) + ";");
```

(continues on next page)

(continued from previous page)

```
    eval("fM2h=fM2." + scores_names(n) + ";");
    eval("fNh=fN." + scores_names(n) + ";");

    [Sih,SiTh] = calcSobolSaltelli(fM1h,fM2h,fNh,stg);

    eval("Si." + scores_names(n) + "=Sih;");
    eval("SiT." + scores_names(n) + "=SiTh;");
end
end

function [Si,SiT] = bootstrap_hq(fM1,fM2,fN,stg,scores_names,j)
for n = 1:size(scores_names,2)
    eval("fM1h=fM1." + scores_names(n) + ";");
    eval("fM2h=fM2." + scores_names(n) + ";");
    eval("fNh=fN." + scores_names(n) + ";");

    rng(j*stg.rseed)
    I=ceil(rand(size(fM1h,1),1)*size(fM1h,1));
    fM1q = fM1h(I,:);
    fM2q = fM2h(I,:);
    fNq = fNh(I,:,:);

    [Sih,SiTh] = calcSobolSaltelli(fM1q,fM2q,fNq,stg);
    eval("Si." + scores_names(n) + "=Sih;");
    eval("SiT." + scores_names(n) + "=SiTh;");
end
end

function [Si,SiT] = calcSobolSaltelli(fM1,fM2,fN,stg)
%Code inspired by Geir Halnes et al. 2009 paper. (Halnes, Geir, et al. J. comp. neuroscience 27.3 (2009): 471.)

[Nsamples,Nvars,Npars]=size(fN);

if(stg.sasubmean) % Makes the model more stable
    fM1 = fM1 - mean(fM1,1);
    fM2 = fM2 - mean(fM2,1);
    for i=1:Npars
        fN(:,:,i) =  fN(:,:,i) - mean(fN(:,:,i),1);
    end
end

EY2 = mean(fM1.*fM2); % Valid definition (see Halnes et. al. Appendix)
VY = sum(fM1.^2)/(Nsamples-1) - EY2;
VYT = sum(fM2.^2)/(Nsamples-1) - EY2;

Si = zeros(Nvars,Npars);
SiT= zeros(Nvars,Npars);

for i=1:Npars
    Si(:,i) = (sum(fM1.*fN(:,:,i))/(Nsamples-1) - EY2)./VY;
    SiT(:,i) = 1 - (sum(fM2.*fN(:,:,i))/(Nsamples-1) - EY2)./VYT;
```

(continues on next page)

(continued from previous page)

```
end
end

function rst = remove_sim_error(rst,stg)
error=[];
error_helper=[];

for n = 1:size(rst.fM1.sd,1)
    if max(rst.fM1.sd(n,:)) == stg.errorscore
        error = [error,n];
    end
    if max(rst.fM2.sd(n,:)) == stg.errorscore
        error = [error,n];
    end
    for m = 1:stg.parnum
        if max(rst.fN.sd(n,:,m)) == stg.errorscore
            error_helper = 1;
        end
    end
    if error_helper == 1
        error = [error,n];
    end
    error_helper = 0;
end

error = unique(error);

if ~isempty(error)
    for n = size(error,2):-1:1
        rst.fM1.sd(error(n),:) = [];
        rst.fM2.sd(error(n),:) = [];
        rst.fN.sd(error(n),:,:) = [];

        rst.fM1.se(error(n),:) = [];
        rst.fM2.se(error(n),:) = [];
        rst.fN.se(error(n),:,:) = [];

        rst.fM1.st(error(n),:) = [];
        rst.fM2.st(error(n),:) = [];
        rst.fN.st(error(n),:,:) = [];

        rst.fM1.xfinal(error(n),:) = [];
        rst.fM2.xfinal(error(n),:) = [];
        rst.fN.xfinal(error(n),:,:) = [];
    end
end
end
```

Takes the matrices *fM1*, *fM2*, and *fN* and calculates sensitivity indexes. It calculates indexes based on the following *oputputs* of the *f_sim_score function*:

- *The scores of each experimental output*
- *The scores of each experiment*

- *The total score*
- *The value of each experimental outputs at the end of the simulation*
- **Inputs** - *fM1*, *fM2*, *fN*, *stg.sasubmean*
- **Outputs** - *SI*, *STI*

Code modified from the Geir Halnes et al. 2009 paper.

References

Halnes, G., Ulfhielm, E., Ljunggren, E.E., Kotaleski, J.H. and Rospars, J.P., 2009. Modelling and sensitivity analysis of the reactions involving receptor, G-protein and effector in vertebrate olfactory receptor neurons. Journal of Computational Neuroscience, 27(3), p.471.

Plots

f_plot

Code

```
function f_plot(rst,stg,mmf)

% Inform the user that the plots are being done
disp("Plotting ...")

data_model = mmf.model.data.data_model;

% Import the data on the first run
load(data_model,'Data','sbtab')

% Generate figure with Scores
if isfield(rst,'diag')
    f_plot_scores(rst.diag,stg,sbtab)
end

% Generate figure with Inputs
if isfield(rst,'diag')
    f_plot_inputs(rst.diag,stg,sbtab)
end

% Generate figure with Outputs
if isfield(rst,'diag')
    f_plot_outputs(rst.diag,stg,sbtab,Data,mmf)
end

% Generate figure with input and Output of all experiments
if isfield(rst,'diag')
    f_plot_in_out(rst.diag,stg,sbtab,Data)
end

% Generate figure with optimization results
```

(continues on next page)

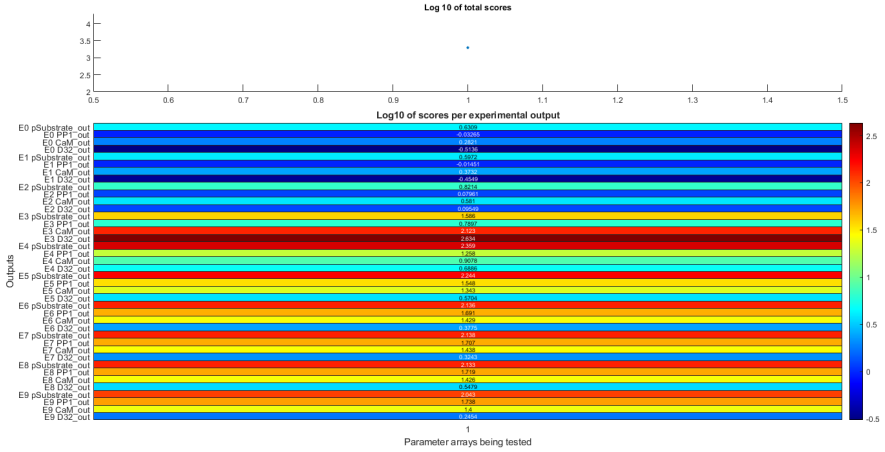
(continued from previous page)

```
if isfield(rst,'opt')
    f_plot_opt(rst,stg)
end

% Generate figures for Sensitivity Analysis
if isfield(rst,'gsa')
    f_plot_gsa_sensitivities(rst.gsa,stg,sbtab);
end
end
```

The function that calls all the custom plot functions when appropriate Plots diagnosis that are important to understand if everything is working as it was supposed, it , expected outputs, observed outputs and scores for the models and conditions specified.

- **Inputs** - *rst*, *stg*
- **Outputs**

Figure Scores



Total scores and scores per dataset given the parameters specified in *stg.pa*

Code Figure Scores

```
function f_plot_scores(rst,stg,sbtab)
set(0,'defaultTextFontName', 'Helvetica')
set(0,'defaultAxesFontName', 'Helvetica')

% Generates a figure with Scores

% Inform the user that fig1 is being ploted
disp("Plotting Scores")

%Closes previous instances of the figure and generates a new one
figHandles = findobj('type', 'figure', 'name', 'Scores');
close(figHandles);
figure('WindowStyle', 'docked','Name','Scores','NumberTitle', 'off');

```

(continues on next page)

(continued from previous page)

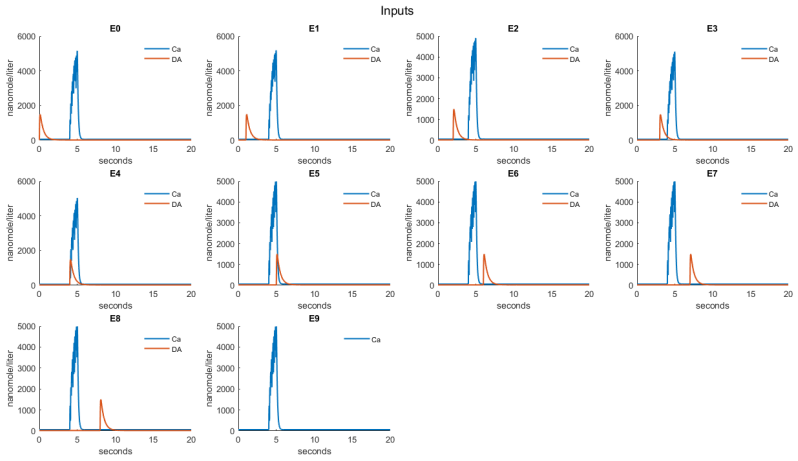
```
% Generate top plot of figure 1
subplot(4,1,1)

% Plot the total scores of each parameter array to test
scatter(stg.pat,[rst(stg.pat).st],20,'filled')
ylabel('Total Score')
set(gca,'xtick',[])
set(gca,'FontSize',10,'Fontweight','bold')

% Choose correct title according to settings
if stg.useLog == 1
    title("Sum of the Log base 10 of the Score of each Experimental␣
↪Output")
elseif stg.useLog == 2
    title("Sum of the Log base 10 of the Score of each Experiment")
elseif stg.useLog == 3
    title("Log base 10 of sum of the Score of all Experiments")
elseif stg.useLog == 4 || stg.useLog == 0
    title("Sum of the Score of all Experiments")
end

% Choose the bounds for the x axis so it aligns with the bottom plot
xlim([min(stg.pat)-0.5 max(stg.pat)+0.5])

% Generate bottom plot of figure 1
subplot(4,1,[2,3,4])

% Generate labels for left of heatmap (Experiment number Dataset␣
↪number)
label = [];

% Iterate over the number of experiments
for n = stg.exprun

    % Iterate over the number of datasets in each experiment
    for j = 1:size(sbtab.datasets(n).output,2)
        label{size(label,2)+1} = {strrep("E" + (n-1) + " " + ...
            string(rst(max(stg.pat)).simd{1,n}.DataNames(...
            end-size(sbtab.datasets(n).output,2)+j)),"_","\_")};
    end
end

% Choose wether to use the score of each dataset or its log base 10
% according to settings
% if stg.useLog == 1 || stg.useLog == 4
heatline = [];

% Iterate over the number of parameter arrays to test
for k = stg.pat
    heatpoint{k} = [];

    % Iterate over the number of experiments
```

(continues on next page)

(continued from previous page)

```
    for n = stg.exprun

        % Get the score of each dataset
        heatpoint{k} = [heatpoint{k},rst(k).sd{n,1}];
    end

    % Combine heatpoints in order to correctly display heatmap
    heatline = [heatline;heatpoint{k}];
end

% Plot the heatmap
h = heatmap(transpose(heatline),'Colormap',turbo,'YDisplayLabels',
↪label,...
    'GridVisible','off','FontSize',10);
h.CellLabelFormat = '%.2e';

title("Score of each Experimental Output")
h.XLabel = 'Parameter arrays being tested';
h.YLabel = 'Experimental Outputs';

end
```

Figure Inputs



Checks inputs to the model

Code Figure Inputs

```
function f_plot_inputs(rst,stg,sbtab)
% Generates a figure with Inputs, one subplot per experiment

% Inform the user that fig2 is being ploted
```

(continues on next page)

(continued from previous page)

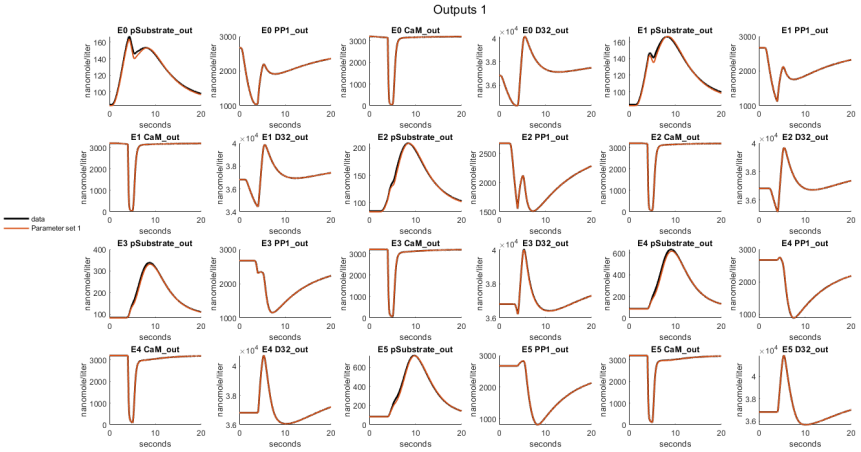
```
disp("Plotting Inputs")

plot_n = 1;
fig_n = 0;
% Iterate over the number of experiments
for n = stg.exprun

    % Generate the right amount of figures for all plots and calculates
    % proper subploting position
    fig_n = f_get_subplot(size(stg.exprun,2),plot_n,fig_n,"Inputs");

    plot_n = plot_n +1;

    hold on

    % Iterate over the number of inputs in each experiment
    for j = 1:size(sbtab.datasets(n).input,2)

        % Iterate over the number of parameter arrays to test
        for m = stg.pat

            % (Until a non broken simulation is found)
            if rst(m).simd{1,n} ~= 0

                % Plot the inputs to each experiment
                plot(rst(m).simd{1,n}.Time,rst(m).simd{1,n}.Data(1:end,...
                    str2double(strrep(sbtab.datasets(n).input(j),'S','))+1),'LineWidth',1.5)

                % Get the correct label for each input of the experiment
                labelfig2(j) = rst(m).simd{1,n}.DataNames(str2double(...
                    strrep(sbtab.datasets(n).input(j),'S',''))+1);

                ylabel(string(rst(m).simd{1,n}.DataInfo{...
                    str2double(strrep(sbtab.datasets(n).input(j),'S','))+1,1}.Units))

                break
            end
        end
    end

    xlabel('seconds')
    % Add a legend to each plot
    legend(labelfig2)
    legend boxoff
    clear labelfig2

```

(continues on next page)

(continued from previous page)

```
    ylim([0 inf])

    % Add a title to each plot
    title("E"+(n-1))

    hold off
end

end
```

Figure Outputs



Expected outputs, observed outputs

Code Figure Outputs

```
function f_plot_outputs(rst,stg,sbtab,Data,mmf)
% Generates a figure with Outputs, one subplot per experimental output

% Inform the user that fig3 is being ploted
disp("Plotting Outputs")

% Get the total number of outputs to set the total number of plots
[plot_tn,~] = f_get_outputs(stg,sbtab);
plot_n = 1;
fig_n = 0;

% Iterate over the number of experiments
for n = stg.exprun

    % Iterate over the number of datasets in each experiment
    for j = 1:size(sbtab.datasets(n).output,2)
```

(continues on next page)

(continued from previous page)

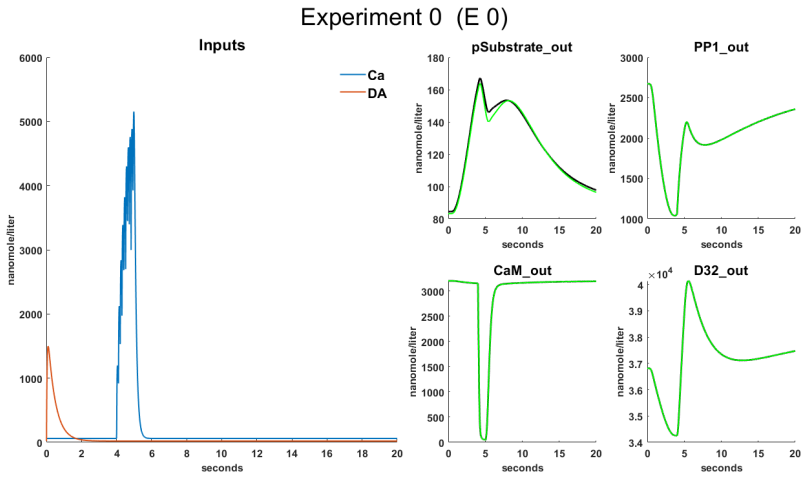
```
        % Generate the right amount of figures for all plots and calculates
        % proper subploting position
        fig_n = f_get_subplot(plot_tn,plot_n,fig_n,"Outputs");

        % Add a legend to the figure
        if mod(plot_n,24) == 1
            Lgnd = legend('show');
            Lgnd.Position(1) = 0;
            Lgnd.Position(2) = 0.5;
            legend boxoff
        end

        plot_n = plot_n + 1;

        hold on

        % Iterate over the number of parameter arrays to test
        for m = stg.pat
            % (Until a non broken simulation is found)
            if rst(m).simd{1,n} ~= 0

                time = rst(m).simd{1,n}.Time;
                data = Data(n).Experiment.x(:,j);

                data_SD = Data(n).Experiment.x_SD(:,j);

                % Plot the outputs to each dataset (new subplots) as they
                % are given in the data provided in sbtab
                scatter(time,data,'filled','k',...
                    'DisplayName','data')

                errorbar(time,data,data_SD, 'vertical',        'k',
↪'LineStyle', 'none','LineWidth',1);

                break
            end
        end

        % Iterate over the number of parameter arrays to test
        for m = stg.pat

            % Plot only if the simulation was successful
            if rst(m).simd{1,n} ~= 0

                time = rst(m).simd{1,n}.Time;
                [sim_results,~] = f_normalize(rst(m),stg,n,j,mmf);
                if stg.simdetail
                    time_detailed = rst(m).simd{1,n+2*stg.expn}.Time;
                    [~,sim_results_detailed]= f_normalize(rst(m),stg,n,j,
↪mmf);
                end
```

(continues on next page)

(continued from previous page)

```
                % Plot the outputs to each dataset (new subplots) and
                % parameter array to test that are simulated using
                % Simbiology
                if stg.simdetail
                    plot(time_detailed,...
                        sim_results_detailed,'DisplayName',...
                        string("Parameter set "+m),'LineWidth',1.5)
                else

                    plot(time,...
                        sim_results,'DisplayName',...
                        string("Parameter set "+m),'LineWidth',1.5)
                end

                ylabel(string(rst(m).simd{1,n}.DataInfo{end-...
                    size(sbtab.datasets(n).output,2)+j,1}.Units))
            end
        end

        hold off

        xlabel('seconds')

        if stg.simdetail
            ylim([min([0,min(sim_results_detailed),min(sim_results),
↪min(data-data_SD),min(data)]) inf])
        else
            ylim([min([0,min(sim_results),min(data-data_SD),min(data)])␣
↪inf])
        end

        % Choose correct title according to settings
        if stg.plotoln == 1
            title("E" + (n-1) + " " +...
                strrep(string(sbtab.datasets(n).output_name{1,j}),'_','\_'))
        else
            title("E" + (n-1) + " " +...
                string(sbtab.datasets(n).output{1,j}))
        end

        % Choose number of decimal places for y axis
        ytickformat('%.2g')
    end
end
end
```

Figure Input and Outputs per experiment



Combined figure of the inputs and outputs for each experiment, on the left side we have the inputs of the experiment and on the right side the outputs

Code Figure Input and Outputs

```
function f_plot_in_out(rst,stg,sbtab,Data)
% Generates a figure with input and Output of all experiments on the left
% side it plots the inputs of the experiment and on the right side it plots
% the outputs

for n = stg.exprun

    helper = 1;
    f_plot_in_out_left(rst,stg,sbtab,helper,...
        size(sbtab.datasets(n).output,2) > 4)

    for j = 1:size(sbtab.datasets(n).output,2)

        if j/4 > helper
            helper = helper +1;
            f_plot_in_out_left(rst,stg,sbtab,helper,...
                size(sbtab.datasets(n).output,2) > 4)
        end

        if size(sbtab.datasets(n).output,2) == 1
            subplot(1,2,j+ceil(j/(2/2))*1)
        elseif size(sbtab.datasets(n).output,2) == 2
            subplot(2,2,j+ceil(j/(2/2))*1)
        elseif size(sbtab.datasets(n).output,2) > 2 &&...
                size(sbtab.datasets(n).output,2) <= 4
            subplot(2,4,j+ceil(j/(4/2))*2)
        elseif size(sbtab.datasets(n).output,2) > 4
            subplot(2,4,j+ceil(j/(4/2))*2-helper*8+8)
        end

        hold on

        % Iterate over the number of parameter arrays to test
```

(continues on next page)

(continued from previous page)

```
        for m = stg.pat
            % (Until a non broken simulation is found)
            if rst(m).simd{1,n} ~= 0

                % Plot the outputs to each dataset (new subplots) as they
                % are given in the data provided in sbtab

                time = rst(m).simd{1,n}.Time;
                data = Data(n).Experiment.x(:,j);
                data_SD = Data(n).Experiment.x_SD(:,j);

                scatter(time,data,'filled','k',...
                    'DisplayName','data')

                errorbar(time,data,data_SD, 'vertical',        'k',
↪'LineStyle', 'none','LineWidth',1);
                break
            end
        end
        % Iterate over the number of parameter arrays to test
        for m = stg.pat

            % Plot only if the simulation was successful
            if rst(m).simd{1,n} ~= 0

                time = rst(m).simd{1,n}.Time;
                [sim_results] = f_normalize(rst(m),stg,n,j);


                if stg.simdetail
                    time_detailed = rst(m).simd{1,n+2*stg.expn}.Time;
                    [~,sim_results_detailed]= f_normalize(rst(m),stg,n,j);
                end

                % Plot the outputs to each dataset (new subplots) and
                % parameter array to test that are simulated using
                % Simbiology
                if stg.simdetail
                    plot(time_detailed,...
                        sim_results_detailed,'DisplayName',...
                        string("Parameter set "+m),'LineWidth',1.5)
                else
                    plot(time,...
                        sim_results,...
                        'DisplayName',string("Parameter set "+m),...
                        'LineWidth',1.5)
                end

                ylabel(string(rst(m).simd{1,n}.DataInfo{end-...
                    size(sbtab.datasets(n).output,2)+j,1}.Units),...
                    'FontSize', 12,'Fontweight','bold')
            end
```

(continues on next page)

(continued from previous page)

```
        end

        hold off

        set(gca,'FontSize',12,'Fontweight','bold')

        if stg.simdetail
            ylim([min([0,min(sim_results_detailed),min(sim_results),
↪min(data-data_SD),min(data)]) inf])
        else
            ylim([min([0,min(sim_results),min(data-data_SD),min(data)]),␣
↪inf])
        end

        xlabel('seconds','FontSize', 12,'Fontweight','bold')

        % Choose correct title according to settings
        if stg.plotoln == 1
            title(strrep(string(sbtab.datasets(n).output_name{1,j}),'_',...
                '\_'),'FontSize', 16,'Fontweight','bold')
        else
            title(string(sbtab.datasets(n).output{1,j}),'FontSize', 16,...
                'Fontweight','bold')
        end

        % Choose number of decimal places for y axis
        ytickformat('%.2g')
    end
end

    function f_plot_in_out_left(rst,stg,sbtab,helper,reuse)
        if reuse
            figHandles = findobj('type', 'figure', 'name', "E " + (n-1) +...
                " " + helper);
            close(figHandles);
            figure('WindowStyle', 'docked','Name', "E " + (n-1)+ " " +...
                helper,'NumberTitle', 'off');
            sgtitle( "Experiment " + (n-1) + " " + helper + "  (E " +...
                (n-1) + " " + helper +")",'FontSize', 28);
        else
            figHandles = findobj('type', 'figure', 'name', "E " + (n-1));
            close(figHandles);
            figure('WindowStyle', 'docked','Name', "E " + (n-1),...
                'NumberTitle', 'off');
            sgtitle( "Experiment " + (n-1) + "  (E " + (n-1) +...
                ")",'FontSize', 28);
        end

        subplot(2,4,[1,2,5,6])

        hold on
        for o = 1:size(sbtab.datasets(n).input,2)
```

(continues on next page)

(continued from previous page)

```
            for p = stg.pat

                % (Until a non broken simulation is found)
                if rst(p).simd{1,n} ~= 0
                    % Plot the inputs to each experiment
                    plot(rst(p).simd{1,n}.Time,rst(p).simd{1,n}.Data(1:end,...
                        str2double(strrep(sbtab.datasets(n).input(o),'S',''
)...
                        )+1),'LineWidth',1.5)

                    % Get the correct label for each input of the
                    % experiment
                    labelfig2(o) = rst(p).simd{1,n}.DataNames(str2double(...
                        strrep(sbtab.datasets(n).input(o),'S',''))+1);

                    ylabel(string(rst(p).simd{1,n}.DataInfo{...
                        str2double(strrep(sbtab.datasets(n).input(o),'S',''
))+1,1}.Units),...
                        'FontSize', 12,'Fontweight','bold')


                    break
                end
            end

        end

        set(gca,'FontSize',12,'Fontweight','bold')

        xlabel('seconds','FontSize', 12,'Fontweight','bold')
        % Add a legend to each plot
        legend(labelfig2,'FontSize', 16,'Fontweight','bold')
        legend boxoff
        clear labelfig2

        ylim([0 inf])

        % Add a title to each plot
        title("Inputs",'FontSize', 18,'Fontweight','bold')

        hold off
    end
end
```

Figure Sensitivity Analysis S_i

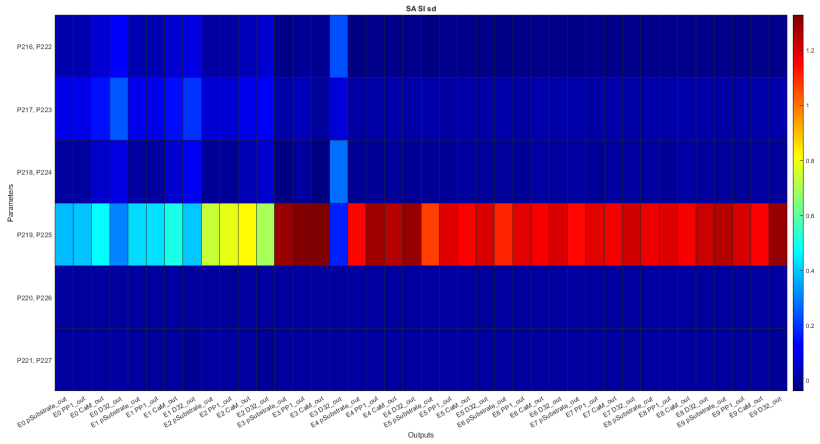
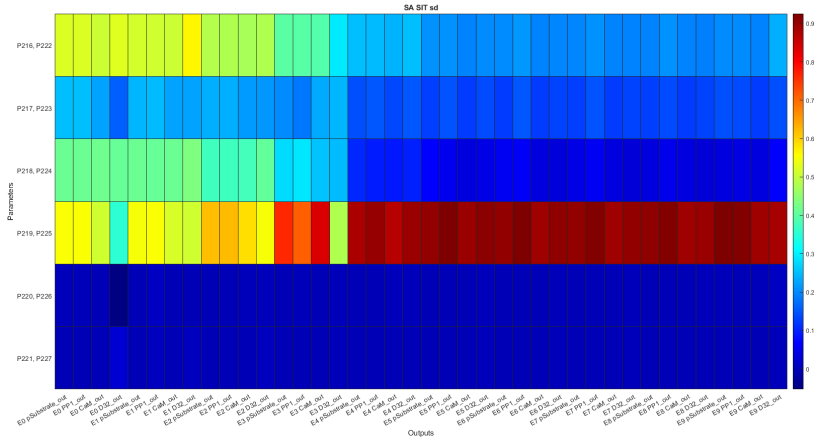


Figure Sensitivity Analysis S_{Ti}



Code figures SA

```
function f_plot_gsa_sensitivities(rst,stg,sbtab)
% Generates figures for Sensitivity Analysis

% Get the total number of outputs
[~,outputNames.sd] = f_get_outputs(stg,sbtab);

for n = 1:size(outputNames.sd,2)
    outputNames.sd{n}{:} = strrep(outputNames.sd{n}{:},"_","\_");
end
for n = stg.exprun
    outputNames.se{n} = "E " + string(n-1);
end

outputNames.xfinal = outputNames.sd;

parNames = cell(1,stg.parnum);
parNames2 = cell(1,stg.parnum);

for n = 1:stg.parnum
    parNames{n} = char("P" + find(stg.partest==n));
```

(continues on next page)

(continued from previous page)

```
end

for n = 1:size(parNames,2)
    parNames2{n} = string(parNames{n}(1,:));
    for m = 2:size(parNames{n},1)
        parNames2{n} = string(parNames2{n}) + ", " +...
            string(parNames{n}(m,:));
    end
end

% Bootstrapping quartile mean of first order Sensitivity index for score
% per Experimental Output
f_generate_plot(rst,stg,outputNames,parNames2,...
    "Si seo bm",...
    "First order Sensitivities calculated using the Score of each␣
↪Experimental Output  (Bootstrapping Mean)",...
    "outputNames.sd",...
    "transpose(reshape(mean(rst.SiQ.sd(:,:,1:stg.parnum)),[size(rst.SiQ.sd,
↪2),size(rst.SiQ.sd,3)]))")

% Bootstrapping quartile mean of total order Sensitivity index for score
% per Experimental Output
f_generate_plot(rst,stg,outputNames,parNames2,"SiT seo bm",...
    "Total order Sensitivities calculated using the Score of each␣
↪Experimental Output (Bootstrapping Mean)",...
    "outputNames.sd",...
    "transpose(reshape(mean(rst.SiTQ.sd(:,:,1:stg.parnum)),[size(rst.SiQ.sd,
↪2),size(rst.SiQ.sd,3)]))")

% Bootstrapping quartile mean of first order Sensitivity index for score
% per Experiments
f_generate_plot(rst,stg,outputNames,parNames2,"Si se bm",...
    "First order Sensitivities calculated using the Score of each␣
↪Experiment(Bootstrapping Mean)",...
    "outputNames.se",...
    "transpose(reshape(mean(rst.SiQ.se(:,:,1:stg.parnum)),[size(rst.SiQ.se,
↪2),size(rst.SiQ.se,3)]))")

% Bootstrapping quartile mean of total order Sensitivity index for score
% per Experiments
f_generate_plot(rst,stg,outputNames,parNames2,"SiT se bm",...
    "Total order Sensitivities calculated using the Score of each␣
↪Experiment (Bootstrapping Mean)",...
    "outputNames.se",...
    "transpose(reshape(mean(rst.SiTQ.se(:,:,1:stg.parnum)),[size(rst.SiQ.se,
↪2),size(rst.SiQ.se,3)]))")

% Bootstrapping quartile mean of first order Sensitivity index for the
% final points of the simulations for the output beeing measured
f_generate_plot(rst,stg,outputNames,parNames2,"Si xfinal bm",...
    "First order Sensitivities calculated using the final value of each␣
↪Experimental Output (Bootstrapping Mean)",...
```

(continues on next page)

(continued from previous page)

```
    "outputNames.xfinal",...
    "transpose(reshape(mean(rst.SiQ.xfinal(:,:,1:stg.parnum)),[size(rst.SiQ.
↪xfinal,2),size(rst.SiQ.xfinal,3)]))")

% Bootstrapping quartile mean of total order Sensitivity index for the
% final points of the simulations for the output beeing measured
f_generate_plot(rst,stg,outputNames,parNames2,"SiT xfinal bm",...
    "Total order Sensitivities calculated using the final value of each␣
↪Experimental Output (Bootstrapping Mean)",...
    "outputNames.xfinal",...
    "transpose(reshape(mean(rst.SiTQ.xfinal(:,:,1:stg.parnum)),[size(rst.
↪SiQ.xfinal,2),size(rst.SiQ.xfinal,3)]))")

figHandles = findobj('type', 'figure', 'name', 'Si,SiT');
close(figHandles);
figure('WindowStyle', 'docked','Name','Si,SiT', 'NumberTitle', 'off');

for n = 1:size(parNames2,2)
    a{n} = char(parNames2{n});
end

a = categorical(a,a);

bar(a,[transpose(rst.Si.st(:,1:stg.parnum)),...
    transpose(rst.SiT.st(:,1:stg.parnum))])
xlabel('Parameters');
ylabel('Sensitivity');
title('Sensitivities calculated using the sum of the Score of all␣
↪Experiments');
legend({'SI','SIT'});
legend boxoff

figHandles = findobj('type', 'figure', 'name', 'Si,SiT b');
close(figHandles);
figure('WindowStyle', 'docked','Name','Si,SiT b', 'NumberTitle', 'off');

T = [];

for n = 1:size(a,2)
    for m = 1:size(rst.SiQ.st(:,n),1)
 T = [T;table(rst.SiQ.st(m,n),a(n),"Si")];
    end
end
for n = 1:size(a,2)
    for m = 1:size(rst.SiTQ.st(:,n),1)
 T = [T;table(rst.SiTQ.st(m,n),a(n),"SiT")];
    end
end

boxchart(T.Var2,T.Var1,'GroupByColor',T.Var3,'MarkerStyle','.',
↪'JitterOutliers','on')
xlabel('Parameters');
```

(continues on next page)

(continued from previous page)

```
ylabel('Sensitivity');
title('Sensitivities calculated using the sum of the Score of all␣
↪Experiments (Bootstrapping)');
legend({'Si','SiT'},'Location','best');
legend boxoff
end

function f_generate_plot(rst,stg,outputNames,parNames2,name,title,...
    helprer1,helprer2)

eval("figHandles = findobj('type', 'figure', 'name', '" + name + "');")
close(figHandles);
eval("figure('WindowStyle', 'docked','Name','" + name +...
    "','NumberTitle', 'off');")

heatmap_fixer = eval(helprer1);
heatmap_fixer=heatmap_fixer(~cellfun('isempty',heatmap_fixer));

heatmap_fixer2 = eval(helprer2);
heatmap_fixer2 = heatmap_fixer2(:,all(~isnan(heatmap_fixer2)));

h = heatmap(heatmap_fixer,parNames2,heatmap_fixer2,'Colormap',turbo,...
    'ColorLimits',[0 1],'GridVisible','off');
h.CellLabelFormat = '%.2f';

eval(" h.Title = """ + title + """;")
h.XLabel = 'Outputs';
h.YLabel = 'Parameters';
end
```

- **Calls**
- **Loads** - *data.mat*

f_get_subplot

Code

```
function fig_n = f_get_subplot(plot_tn,plot_n,fig_n,fig_name)

size_x = 4;
size_y = 6;
size_t = 24;
ratio_1 = 2;
ratio_2 = 3;

% If the amount of plots is bigger thatn the maximum amount of plots␣
↪per
% figure subdivide the plots to more than one figure
if plot_tn > 24

    % Generate a new figure for the first plot and each time the␣
↪number of
```

(continues on next page)

(continued from previous page)

```
    % plots is greater than figure number divided by max plot number␣
↪per
    % figure
    if mod(plot_n-1,24) == 0

        fig_n = fig_n + 1;

        %Close previous instances of the figure and generates a new one
        figHandles = findobj('type', 'figure', 'name', fig_name + " "␣
↪+ fig_n);
        close(figHandles);
        figure('WindowStyle', 'docked','Name', fig_name + " " + fig_n,
↪'NumberTitle', 'off');
        sgtitle(fig_name + " " + fig_n);
    end

    % Get the correct subploting position for each plot
    if plot_tn/24 < fig_n
        subplot(ceil(sqrt((plot_tn-(fig_n-1)*24)/6)*2),ceil(sqrt((plot_
↪tn-(fig_n-1)*24)/6)*3),plot_n-(fig_n-1)*24)
    else
        subplot(ceil(sqrt(24/6)*2),ceil(sqrt(24/6)*3),plot_n-(fig_n-
↪1)*24)
    end

else

    % Generate a new figure for the first plot
    if mod(plot_n-1,24) == 0

        %Close previous instances of the figure and generates a new one
        figHandles = findobj('type', 'figure', 'name', fig_name);
        close(figHandles);
        figure('WindowStyle', 'docked','Name', fig_name, 'NumberTitle',
↪ 'off');
        sgtitle(fig_name);

    end

    % Get the correct subploting position for each plot
    subplot(ceil(sqrt(plot_tn/6)*2),ceil(sqrt(plot_tn/6)*3),plot_n)

end
end
```

- **Inputs**
- **Outputs**
- **Calls**
- **Loads**

General purpose

f_save_analysis

Code

```
function f_save_analysis(stg,sb,rst,mmf)

Results_Folder = mmf.model.results.main;
Analysis_folder = mmf.model.results.analysis.main;
Analysis_date_folder = mmf.model.results.analysis.date.main;

[~,~] = mkdir(Results_Folder);
[~,~] = mkdir(Analysis_folder);
[~,~] = mkdir(Analysis_date_folder);
addpath(Analysis_date_folder)

save (Analysis_date_folder + "Analysis.mat",'stg','sb','rst');
end
```

- **Inputs**
- **Outputs**
- **Calls**
- **Loads**

f_save_plots

Code

```
function f_save_plots(mmf)

Analysis_date_folder = mmf.model.results.analysis.date.main;

FigList = findobj(allchild(0), 'flat', 'Type', 'figure');

[~,~] = mkdir(Analysis_date_folder);

savefig(FigList(end:-1:1),...
    Analysis_date_folder + "All_figures.fig");

for iFig = 1:length(FigList)
    FigHandle = FigList(iFig);
    FigName   = get(FigHandle, 'Name');

    saveas(FigHandle, Analysis_date_folder + FigName + ".png")
end
end
```

- **Inputs**
- **Outputs**
- **Calls**

- **Loads**

f_get_outputs

Code

```
function [nOutputs,outputNames] = f_get_outputs(stg,sbtab)

persistent n_out
persistent out_name

if isempty(n_out)
    n_out = 0;
    out_name = [];
    for n = stg.exprun
        for j = 1:size(sbtab.datasets(n).output,2)
            n_out = n_out + 1;
            out_name{n_out} = {"E" + (n-1) + " " + string(sbtab.datasets(n).output{1,j})};
        end
    end
end

nOutputs = n_out;
outputNames = out_name;
end
```

- **Inputs**
- **Outputs**
- **Calls**
- **Loads**

3.2.6 Settings file

A place for the user to define all the relevant properties of model simulation that are not stored in SBtab. This are usually things that need to change during optimizations or model development.

These settings files can be found can be found on the respective model repository in the directory "Matlab/Settings", in the example model from our main repository in the directory "Matlab/model/Model_Example/Matlab/Settings", or by following these links:

- Example model settings files
- Fujita_2010 model settings files
- Nair_2016 model settings files
- Viswan_2018 model settings files

Default settings code

```
function [stg] = default_settings()

%% Import

% True or false to decide whether to run import functions (Import)
stg.import = true;


% Name of the excel file with the sbtab (SBtab excel name)
stg.sbtab_excel_name = "SBTAB.xlsx";

% Name of the model (Name)
stg.name = "model_name";

% Name of the default model compartment (Compartment name)
stg.cname = "Compartment";

% Name of the sbtab saved in .mat format (SBtab name)
stg.sbtab_name = "SBtab_" + stg.name;

%% Analysis

% Experiments to run (example experiment 1 to 3 and experimet 6)
stg.exprun = [1:3,6];

% Choice between 0,1,2 and 3,4 to choose the scoring method (check
% documentation) (Use logarithm)
stg.useLog = 4;

% True or false to decide whether to use multicore everywhere it is
% available (Optimization Multicore)
stg.optmc = true;

% Choice of ramdom seed (Ramdom seed)
stg.rseed = 1;

% True or false to decide whether to use display simulation diagnostics in
% the console (Simulation Console)
stg.simcsl = false;

% True or false to decide whether to display optimization results on
% console (Optimization console)
stg.optcsl = false;

% True or false to decide whether to save results (Save results)
stg.save_results = true;

% True or false to decide whether to run detailed simulation for plots
stg.simdetail = true;

%% Simulation

% Maximum time for each individual function to run in seconds (Maximum
```

(continues on next page)

(continued from previous page)

```
% time)
stg.maxt = 10;

% Equilibration time (Equilibration time)
stg.eqt  = 50000;

% True or false to decide whether to do Dimensional Analysis (Dimensional
% Analysis)
stg.dimenanal = false;

% True or false to decide whether to do Unit conversion (Unit conversion)
stg.UnitConversion = false;

% True or false to decide whether to do Absolute Tolerance Scaling
% (Absolute Tolerance Scaling)
stg.abstolscale = true;

% Value of Relative tolerance (Relative tolerance)
stg.reltol = 1.0E-4;

% Value of Absolute tolerance (Absolute tolerance)
stg.abstol = 1.0E-4;

% Time units for simulation (Simulation time)
stg.simtime = "second";

% True or false to decide whether to run sbioaccelerate (after changing
% this value you need to run "clear functions" to see an effect)
% (sbioaccelerate)
stg.sbioacc = true;

% (Absolute tolerance step size for equilibration)
stg.abstolstepsize_eq = [];

% Max step size in the simulation (if empty matlab decides whats best)
% (Maximum step)
stg.maxstep = [10];

% Max step size in the equilibration (if empty matlab decides whats best)
% (Maximum step)
stg.maxstepeq = [2];

% Max step size in the detailed plots (if empty matlab decides whats best)
% (Maximum step)
stg.maxstepdetail = [2];

% Default score when there is a simulation error, this is needed to keep
% the optimizations working.
% (error score)
stg.errorscore = 10^5;
%% Model

```

(continues on next page)

(continued from previous page)

```
% Number of parameters to optimize (Parameter number)
stg.parnum = 5;

original_parameter_set = zeros(1,10);

% Array with the lower bound of all parameters (Lower bound)
stg.lb = original_parameter_set-5;

% Array with the upper bound of all parameters (Upper bound)
stg.ub = original_parameter_set+5;

%% Diagnostics

% Choice of what parameters in the array to test, the indices correspond to
% the parameters in the model and the numbers correspond to the parameters
% in the optimization array, usually not all parameters are optimized so
% there needs to be a match between one and the other. (Parameters to test)
% In this example there are ten parameters in this imaginary model and we
% are only interested in parameter 2,4,8,9, and 10. Note that stg.parnum is
% five because of this and not ten
stg.partest(:,1) = [0,1,0,2,0,0,0,3,4,5];

% (Parameter array to test)
stg.pat = [1:2];

% All the parameter arrays, in this case there is only one (parameters here
% are in log10 space)(Parameter arrays)
stg.pa(1,:) = [1,1,1,1,1];
stg.pa(1,:) = [1,0,1,2,1];

% Best parameter array found so far for the model (Best parameter array)
stg.bestpa = stg.pa(1,:);

%% Plots

% True or false to decide whether to plot results (Plots)
stg.plot = true;

% True or false to decide whether to use long names in the title of the
% outputs plots in f_plot_outputs.m (Plot outputs long names)
stg.plotoln = true;

%% Sensitivity analysis

% Number of samples to use in SA (Sensitivity analysis number of samples)
stg.sansamples = 100;

% True or false to decide whether to subtract the mean before calculating
% SI and SIT (Sensitivity analysis subtract mean)
stg.sasubmean = true;

% Choose the way you want to obtain the samples of the parameters for
```

(continues on next page)

(continued from previous page)

```
% performing the SA; 0 Log uniform distribution truncated at the parameter
% bounds 1 Log normal distribution with mu as the best value for a
% parameter and sigma as stg.sasamplesigma truncated at the parameter
% bounds 2 same as 1 without truncation 3 Log normal distribution centered
% at the mean of the parameter bounds and sigma as stg.sasamplesigma
% truncated at the parameter bounds 4 same as 3 without truncation.
% (Sensitivity analysis sampling mode)
stg.sasamplemode = 2;

% Sigma for creating the normal distribution of parameters to perform
% sensitivity analysis (Sensitivity analysis sampling sigma)
stg.sasamplesigma = 0.1;

%% Optimization

%  Time for the optimization in seconds (fmincon does not respect this
% time!!) (Optimization time)
stg.optt = 60*5;

% Population size for the algorithms that use populations (Population size)
stg.popsize = 144;

% optimization start method, choose between: 1 Random starting point or
% group of starting points inside the bounds 2 Random starting point or
% group of starting points near the best point (Optimization start method)
stg.osm = 1;

% Distance from best parameter array to be used in stg.osm method 2
% (Distance from best parameter array)
stg.dbs = 0.1;

% True or false to decide whether to use Multistart (Multistart)
stg.mst = false;

% Multistart size
stg.msts = 1;

% True or false to decide whether to display Plots (Plots doesn't work if
% using multicore) (Optimization plots)
stg.optplots = true;

% True or false to decide whether to run fmincon (no gradient so this
% doesn't work very well, no max time!!)
stg.fmincon = false;

% Options for fmincon (fmincon options)
stg.fm_options = optimoptions('fmincon',...
    'Algorithm','interior-point',...
    'MaxIterations',2,'OptimalityTolerance',0,...
    'StepTolerance',1e-6,'FiniteDifferenceType','central',...
    'MaxFunctionEvaluations',10000);

```

(continues on next page)

(continued from previous page)

```
% True or false to decide whether to run simulated annealing (Simulated
% annealing)
stg.sa = false;

% Options for simulated annealing (Simulated annealing options)
stg.sa_options = optimoptions(@simulannealbnd, ...
    'MaxTime',stg.optt,...
    'ReannealInterval',40);

% True or false to decide whether to run Pattern search (Pattern search)
stg.psearch = false;

% Options for Pattern search (Pattern search options)
stg.psearch_options = optimoptions(@patternsearch,...
    'MaxTime',stg.optt,'UseParallel',stg.optmc,...
    'UseCompletePoll',true,'UseCompleteSearch',true,...
    'MaxMeshSize',2,'MaxFunctionEvaluations',2000);

% True or false to decide whether to run Genetic algorithm (Genetic
% algorithm)
stg.ga = false;

% Options for Genetic algorithm (Genetic algorithm options)
stg.ga_options = optimoptions(@ga,'MaxGenerations',200,...
    'MaxTime',stg.optt,'UseParallel',stg.optmc,...
    'PopulationSize',stg.popsize,...
    'MutationFcn','mutationadaptfeasible');

% True or false to decide whether to run Particle swarm (Particle swarm)
stg.pswarm = false;

% Options for Particle swarm (Particle swarm options)
stg.pswarm_options = optimoptions('particleswarm',...
    'MaxTime',stg.optt,'UseParallel',stg.optmc,'MaxIterations',200,...
    'SwarmSize',stg.popsize);

% True or false to decide whether to run Surrogate optimization (Surrogate
% optimization)
stg.sopt = false;

% Options for Surrogate optimization (Surrogate optimization options)
stg.sopt_options = optimoptions('surrogateopt',...
    'MaxTime',stg.optt,'UseParallel',stg.optmc,...
    'MaxFunctionEvaluations',5000,...
    'MinSampleDistance',0.2,'MinSurrogatePoints',32*2+1);
end
```

Example settings code

```
function [stg] = Example_model()

%% Import
```

(continues on next page)

(continued from previous page)

```
% True or false to decide whether to run import functions (Import)
stg.import = true;

% Name of the excel file with the sbtab (SBtab excel name)
stg.sbtab_excel_name = "SBTAB example.xlsx";

% Name of the model (Name)
stg.name = "Example";

% Name of the default model compartment (Compartment name)
stg.cname = "Compartment";

%% Analysis

% Experiments to run
stg.exprun = [1,3];
% stg.exprun = [1,2,3];

% Choice between 0,1,2 and 3 to change either and how to apply log10 to the
% scores (check documentation) (Use logarithm)
stg.useLog = 0;

% True or false to decide whether to use multicore everywhere it is
% available (Optimization Multicore)
stg.optmc = false;

% Choice of ramdom seed (Ramdom seed)
stg.rseed = 1;

% True or false to decide whether to use display simulation diagnostics in
% the console (Simulation Console)
stg.simcsl = false;

% True or false to decide whether to display optimization results on
% console (Optimization console)
stg.optcsl = true;

% True or false to decide whether to display PLA results on console (PLA
% console)
stg.placsl = true;

% True or false to decide whether to save results (Save results)
stg.save_results = true;

% True or false to decide whether to run detailed simulation for plots
stg.simdetail = false;

%% Simulation

% Maximum time for each individual function to run in seconds (Maximum
% time)
```

(continues on next page)

(continued from previous page)

```
stg.maxt = 2;

% Equilibration time (Equilibration time)
stg.eqt  = 50000;

% True or false to decide whether to do Dimensional Analysis (Dimensional
% Analysis)
stg.dimenanal = true;

% True or false to decide whether to do Unit conversion (Unit conversion)
stg.UnitConversion = true;

% True or false to decide whether to do Absolute Tolerance Scaling
% (Absolute Tolerance Scaling)
stg.abstolscale = true;

% Value of Relative tolerance (Relative tolerance)
stg.reltol = 1.0E-4;

% Value of Absolute tolerance (Absolute tolerance)
stg.abstol = 1.0E-7;

% Time units for simulation (Simulation time)
stg.simtime = "second";

% True or false to decide whether to run sbioaccelerate (after changing
% this value you need to run "clear functions" to see an effect)
% (sbioaccelerate)
stg.sbioacc = false;

% Max step size in the simulation (if empty matlab decides whats best)
% (Maximum step)
stg.maxstep = [];

% Max step size in the equilibration (if empty matlab decides whats best)
% (Maximum step)
stg.maxstepeq = [];

% Max step size in the detailed plots (if empty matlab decides whats best)
% (Maximum step)
stg.maxstepdetail = [0.001];

% Default score when there is a simulation error, this is needed to keep
% the optimizations working. (error score)
stg.errorscore = 10^5;

%% Model

% Number of parameters to optimize (Parameter number)
stg.parnum = 12;

% Index for the parameters that have thermodynamic constrains (Termodiamic
```

(continues on next page)

(continued from previous page)

```
% Constrains Index)
stg.tci = [8];

% Parameters to multiply to the first parameter (in Stg.partest to get to
% the correct thermodynamic constrain formula) (Termodiamic Constrains
% multipliers)
stg.tcm([8],1) = [4];
stg.tcm([8],2) = [5];
stg.tcm([8],3) = [7];

% Parameters to divide to the first parameter (in Stg.partest to get to the
% correct thermodynamic constrain formula) (Termodiamic Constrains
% divisors)
stg.tcd([8],1) = [1];
stg.tcd([8],2) = [3];
stg.tcd([8],3) = [6];

% Array with the lower bound of all parameters (Lower bound)
stg.lb = zeros(1,stg.parnum)-4;

% Array with the upper bound of all parameters (Upper bound)
stg.ub = zeros(1,stg.parnum)+4;

%% Diagnostics

% Choice of what parameters in the array to test, the indices correspond to
% the parameters in the model and the numbers correspond to the parameters
% in the optimization array, usually not all parameters are optimized so
% there needs to be a match between one and the other. (Parameters to test)
stg.partest(:,1) = [1  ,2  ,3  ,4  ,5  ,6  ,7 ,2 ,8 ,9,...
    10 ,11 ,12];

% (Parameter array to test)
stg.pat = 1:3;

% All the parameter arrays, in this case there is only one (Parameter
% arrays)
stg.pa(1,:) = [3.999,0.696,1.072,3.429,-0.751,-3.741,-0.569,0.831,3.068,0.921,-
↪2.156,-1.970];
stg.pa(2,:) = stg.pa(1,:)-1;
stg.pa(3,:) = stg.pa(1,:)+1;

% Best parameter array found so far for the model (Best parameter array)
stg.bestpa = stg.pa(1,:);

%% Plots

% True or false to decide whether to plot results (Plots)
stg.plot = true;

% True or false to decide whether to use long names in the title of the
% outputs plots in f_plot_outputs.m (Plot outputs long names)
```

(continues on next page)

(continued from previous page)

```
stg.plotnames = true;

%% Sensitivity analysis

% Number of samples to use in SA (Sensitivity analysis number of samples)
stg.sansamples = 1000;

% True or false to decide whether to subtract the mean before calculating
% SI and SIT (Sensitivity analysis subtract mean)
stg.sasubmean = true;

% Choose the way you want to obtain the samples of the parameters for
% performing the SA; 0 Log uniform distribution truncated at the parameter
% bounds 1 Log normal distribution with mu as the best value for a
% parameter and sigma as stg.sasamplesigma truncated at the parameter
% bounds 2 same as 1 without truncation 3 Log normal distribution centered
% at the mean of the parameter bounds and sigma as stg.sasamplesigma
% truncated at the parameter bounds 4 same as 3 without truncation.
% (Sensitivity analysis sampling mode)
stg.sasamplemode = 2;

% Sigma for creating the normal distribution of parameters to perform
% sensitivity analysis (Sensitivity analysis sampling sigma)
stg.sasamplesigma = 0.1;

stg.gsabootstrapsize = ceil(sqrt(stg.sansamples));

%% Optimization

%  Time for the optimization in seconds (fmincon does not respect this
% time!!) (Optimization time)
stg.optt = 60*1;

% Population size for the algorithms that use populations (Population size)
stg.popsize = 10;

% optimization start method, choose between: 1 Random starting point or
% group of starting points inside the bounds 2 Random starting point or
% group of starting points near the best point (Optimization start method)
stg.osm = 1;

% Distance from best parameter array to be used in stg.osm method 2
% (Distance from best parameter array)
stg.dbs = 0.1;

% True or false to decide whether to use Multistart (Multistart)
stg.mst = false;

% Multistart size
stg.msts = 1;

% True or false to decide whether to display Plots (Plots doesn't work if
```

(continues on next page)

(continued from previous page)

```
% using multicore) (Optimization plots)
stg.optplots = true;

% True or false to decide whether to run fmincon (no gradient so this
% doesn't work very well, no max time!!)
stg.fmincon = false;

% Options for fmincon (fmincon options)
stg.fm_options = optimoptions('fmincon',...
    'UseParallel',stg.optmc,...
    'Algorithm','interior-point',...
    'MaxIterations',2,'OptimalityTolerance',0,...
    'StepTolerance',1e-6,'FiniteDifferenceType','central',...
    'MaxFunctionEvaluations',10000);

% True or false to decide whether to run simulated annealing (Simulated
% annealing)
stg.sa = false;

% Options for simulated annealing (Simulated annealing options)
stg.sa_options = optimoptions(@simulannealbnd, ...
    'MaxTime',stg.optt,...
    'InitialTemperature',...
    ones(1,stg.parnum)*2,'ReannealInterval',40);

% True or false to decide whether to run Pattern search (Pattern search)
stg.psearch = false;

% Options for Pattern search (Pattern search options)
stg.psearch_options = optimoptions(@patternsearch,...
    'MaxTime',stg.optt,'UseParallel',stg.optmc,...
    'UseCompletePoll',true,'UseCompleteSearch',true,...
    'MaxMeshSize',2,'MaxFunctionEvaluations',2000);

% True or false to decide whether to run Genetic algorithm (Genetic
% algorithm)
stg.ga = true;

% Options for Genetic algorithm (Genetic algorithm options)
stg.ga_options = optimoptions(@ga,'MaxGenerations',200,...
    'MaxTime',stg.optt,'UseParallel',stg.optmc,...
    'PopulationSize',stg.popsize,...
    'MutationFcn','mutationadaptfeasible','Display','diagnose');

% True or false to decide whether to run Particle swarm (Particle swarm)
stg.pswarm = false;

% Options for Particle swarm (Particle swarm options)
stg.pswarm_options = optimoptions('particleswarm',...
    'MaxTime',stg.optt,'UseParallel',stg.optmc,...
    'SwarmSize',stg.popsize);

```

(continues on next page)

(continued from previous page)

```
% True or false to decide whether to run Surrogate optimization (Surrogate
% optimization)
stg.sopt = false;

% Options for Surrogate optimization (Surrogate optimization options)
stg.sopt_options = optimoptions('surrogateopt',...
    'MaxTime',stg.optt,'UseParallel',stg.optmc,...
    'MaxFunctionEvaluations',5000,...
    'MinSampleDistance',0.2,'MinSurrogatePoints',32*2+1);
end
```

Import

Default settings code

```
% True or false to decide whether to run import functions (Import)
stg.import = true;


% Name of the excel file with the sbtab (SBtab excel name)
stg.sbtab_excel_name = "SBTAB.xlsx";

% Name of the model (Name)
stg.name = "model_name";

% Name of the default model compartment (Compartment name)
stg.cname = "Compartment";

% Name of the sbtab saved in .mat format (SBtab name)
stg.sbtab_name = "SBtab_" + stg.name;
```

Example settings code

```
% True or false to decide whether to run import functions (Import)
stg.import = true;

% Name of the excel file with the sbtab (SBtab excel name)
stg.sbtab_excel_name = "SBTAB example.xlsx";

% Name of the model (Name)
stg.name = "Example";

% Name of the default model compartment (Compartment name)
stg.cname = "Compartment";
```

- **stg.import** - (logical) Decide whether to run import functions
- **stg.sbtab_excel_name** - (string) Name of the Excel file with the SBtab
- **stg.name** - (string) Name of the model
- **stg.cname** - (string) Name of the default model compartment
- **stg.sbtab_name** - (string) Name of the SBtab saved in .mat format

Analysis

Default settings code

```
% Experiments to run (example experiment 1 to 3 and experimet 6)
stg.exprun = [1:3,6];

% Choice between 0,1,2 and 3,4 to choose the scoring method (check
% documentation) (Use logarithm)
stg.useLog = 4;

% True or false to decide whether to use multicore everywhere it is
% available (Optimization Multicore)
stg.optmc = true;

% Choice of ramdom seed (Ramdom seed)
stg.rseed = 1;

% True or false to decide whether to use display simulation diagnostics in
% the console (Simulation Console)
stg.simcsl = false;

% True or false to decide whether to display optimization results on
% console (Optimization console)
stg.optcsl = false;

% True or false to decide whether to save results (Save results)
stg.save_results = true;

% True or false to decide whether to run detailed simulation for plots
stg.simdetail = true;
```

Example settings code

```
% Experiments to run
stg.exprun = [1,3];
% stg.exprun = [1,2,3];

% Choice between 0,1,2 and 3 to change either and how to apply log10 to the
% scores (check documentation) (Use logarithm)
stg.useLog = 0;

% True or false to decide whether to use multicore everywhere it is
% available (Optimization Multicore)
stg.optmc = false;

% Choice of ramdom seed (Ramdom seed)
stg.rseed = 1;

% True or false to decide whether to use display simulation diagnostics in
% the console (Simulation Console)
stg.simcsl = false;

% True or false to decide whether to display optimization results on
```

(continues on next page)

(continued from previous page)

```
% console (Optimization console)
stg.optcsl = true;

% True or false to decide whether to display PLA results on console (PLA
% console)
stg.placsl = true;

% True or false to decide whether to save results (Save results)
stg.save_results = true;

% True or false to decide whether to run detailed simulation for plots
stg.simdetail = false;
```

- **stg.exprun** - (double) Experiments to run
- **stg.useLog** - (double) Choice between 0,1,2 and 3 to change either and how to apply log10 to the scores, check *results*:
- **stg.optmc** - (logical) Decide whether to use multicore everywhere it is available
- **stg.rseed** - (double) Choice of random seed
- **stg.simcsl** - (logical) Decide whether to display simulation diagnostics in the console
- **stg.optcsl** - (logical) Decide whether to display optimization results on console
- **stg.save_results** - (logical) Decide whether to save results
- **stg.simdetail** - (logical) Decide whether to run detailed simulation for plots

Simulation

Default settings code

```
% Maximum time for each individual function to run in seconds (Maximum
% time)
stg.maxt = 10;

% Equilibration time (Equilibration time)
stg.eqt  = 50000;

% True or false to decide whether to do Dimensional Analysis (Dimensional
% Analysis)
stg.dimenanal = false;

% True or false to decide whether to do Unit conversion (Unit conversion)
stg.UnitConversion = false;

% True or false to decide whether to do Absolute Tolerance Scaling
% (Absolute Tolerance Scaling)
stg.abstolscale = true;

% Value of Relative tolerance (Relative tolerance)
stg.reltol = 1.0E-4;

```

(continues on next page)

(continued from previous page)

```
% Value of Absolute tolerance (Absolute tolerance)
stg.abstol = 1.0E-4;

% Time units for simulation (Simulation time)
stg.simtime = "second";

% True or false to decide whether to run sbioaccelerate (after changing
% this value you need to run "clear functions" to see an effect)
% (sbioaccelerate)
stg.sbioacc = true;

% (Absolute tolerance step size for equilibration)
stg.abstolstepsize_eq = [];

% Max step size in the simulation (if empty matlab decides whats best)
% (Maximum step)
stg.maxstep = [10];

% Max step size in the equilibration (if empty matlab decides whats best)
% (Maximum step)
stg.maxstepeq = [2];

% Max step size in the detailed plots (if empty matlab decides whats best)
% (Maximum step)
stg.maxstepdetail = [2];

% Default score when there is a simulation error, this is needed to keep
% the optimizations working.
% (error score)
stg.errorscore = 10^5;
```

Example settings code

```
% Maximum time for each individual function to run in seconds (Maximum
% time)
stg.maxt = 2;

% Equilibration time (Equilibration time)
stg.eqt  = 50000;

% True or false to decide whether to do Dimensional Analysis (Dimensional
% Analysis)
stg.dimenanal = true;

% True or false to decide whether to do Unit conversion (Unit conversion)
stg.UnitConversion = true;

% True or false to decide whether to do Absolute Tolerance Scaling
% (Absolute Tolerance Scaling)
stg.abstolscale = true;

% Value of Relative tolerance (Relative tolerance)
```

(continues on next page)

(continued from previous page)

```
stg.reltol = 1.0E-4;

% Value of Absolute tolerance (Absolute tolerance)
stg.abstol = 1.0E-7;

% Time units for simulation (Simulation time)
stg.simtime = "second";

% True or false to decide whether to run sbioaccelerate (after changing
% this value you need to run "clear functions" to see an effect)
% (sbioaccelerate)
stg.sbioacc = false;

% Max step size in the simulation (if empty matlab decides whats best)
% (Maximum step)
stg.maxstep = [];

% Max step size in the equilibration (if empty matlab decides whats best)
% (Maximum step)
stg.maxstepeq = [];

% Max step size in the detailed plots (if empty matlab decides whats best)
% (Maximum step)
stg.maxstepdetail = [0.001];

% Default score when there is a simulation error, this is needed to keep
% the optimizations working. (error score)
stg.errorscore = 10^5;
```

- **stg.maxt** - (double) Maximum time for each individual function has to run in seconds
- **stg.eqt** - (double) Equilibration time in seconds
- **stg.dimenanal** - (logical) Decide whether to do Dimensional Analysis
- **stg.UnitConversion** - (logical) Decide whether to do Unit conversion
- **stg.abstolscale** - (logical) Decide whether to do Absolute Tolerance Scaling
- **stg.reltol** - (double) Value of Relative tolerance
- **stg.abstol** - (double) Value of Absolute tolerance
- **stg.simtime** - (string) Time units for simulation
- **stg.sbioacc** - (logical) Decide whether to run sbioaccelerate (after changing this value you need to run "clear functions" to see an effect)
- **stg.abstolstepsize_eq** - (double) Absolute tolerance step size for equilibration (if empty MATLAB® decides whats best)
- **stg.maxstep** - (double) Max step size in the simulation (if empty MATLAB® decides what's best)
- **stg.maxstepeq** - (double) Max step size in the equilibration (if empty MATLAB® decides whats best)
- **stg.maxstepdetail** - (double) Max step size in the detailed plots (if empty MATLAB® decides whats best)
- **stg.errorscore** - (double) Default score when there is a simulation error, this is needed to keep the optimizations working.

Model

Default settings code

```
% Number of parameters to optimize (Parameter number)
stg.parnum = 5;

original_parameter_set = zeros(1,10);

% Array with the lower bound of all parameters (Lower bound)
stg.lb = original_parameter_set-5;

% Array with the upper bound of all parameters (Upper bound)
stg.ub = original_parameter_set+5;
```

Example settings code

```
% Number of parameters to optimize (Parameter number)
stg.parnum = 12;

% Index for the parameters that have thermodynamic constrains (Termodiamic
% Constrains Index)
stg.tci = [8];

% Parameters to multiply to the first parameter (in Stg.partest to get to
% the correct thermodynamic constrain formula) (Termodiamic Constrains
% multipliers)
stg.tcm([8],1) = [4];
stg.tcm([8],2) = [5];
stg.tcm([8],3) = [7];

% Parameters to divide to the first parameter (in Stg.partest to get to the
% correct thermodynamic constrain formula) (Termodiamic Constrains
% divisors)
stg.tcd([8],1) = [1];
stg.tcd([8],2) = [3];
stg.tcd([8],3) = [6];

% Array with the lower bound of all parameters (Lower bound)
stg.lb = zeros(1,stg.parnum)-4;

% Array with the upper bound of all parameters (Upper bound)
stg.ub = zeros(1,stg.parnum)+4;
```

- **stg.parnum** - (double) Number of parameters to optimize
- **stg.tci** - (double) Index for the parameters that have thermodynamic constraints
- **stg.tcm** - (double) Parameters to multiply to the first parameter (in *stg.partest* to get to the correct thermodynamic constraint formula)
- **stg.tcd** - (double) Parameters to divide to the first parameter (in *stg.partest* to get to the correct thermodynamic constraint formula)
- **stg.lb** - (double) Lower bound of all parameters

$$stg.lb = \begin{bmatrix} lb_1 & lb_2 & ... & lb_i \end{bmatrix}$$

- i = Parameter index

- **stg.ub** - (double) Upper bound of all parameters

$$stg.up = \begin{bmatrix} ub_1 & ub_2 & ... & ub_i \end{bmatrix}$$

  - i = Parameter index

Diagnostics

Default settings code

```
% Choice of what parameters in the array to test, the indices correspond to
% the parameters in the model and the numbers correspond to the parameters
% in the optimization array, usually not all parameters are optimized so
% there needs to be a match between one and the other. (Parameters to test)
% In this example there are ten parameters in this imaginary model and we
% are only interested in parameter 2,4,8,9, and 10. Note that stg.parnum is
% five because of this and not ten
stg.partest(:,1) = [0,1,0,2,0,0,0,3,4,5];

% (Parameter array to test)
stg.pat = [1:2];

% All the parameter arrays, in this case there is only one (parameters here
% are in log10 space)(Parameter arrays)
stg.pa(1,:) = [1,1,1,1,1];
stg.pa(1,:) = [1,0,1,2,1];

% Best parameter array found so far for the model (Best parameter array)
stg.bestpa = stg.pa(1,:);
```

Example settings code

```
% Choice of what parameters in the array to test, the indices correspond to
% the parameters in the model and the numbers correspond to the parameters
% in the optimization array, usually not all parameters are optimized so
% there needs to be a match between one and the other. (Parameters to test)
stg.partest(:,1) = [1  ,2  ,3  ,4  ,5  ,6  ,7 ,2 ,8 ,9,...
    10 ,11 ,12];

% (Parameter array to test)
stg.pat = 1:3;

% All the parameter arrays, in this case there is only one (Parameter
% arrays)
stg.pa(1,:) = [3.999,0.696,1.072,3.429,-0.751,-3.741,-0.569,0.831,3.068,0.921,-
↪2.156,-1.970];
stg.pa(2,:) = stg.pa(1,:)-1;
stg.pa(3,:) = stg.pa(1,:)+1;

% Best parameter array found so far for the model (Best parameter array)
stg.bestpa = stg.pa(1,:);
```

- **stg.partest** - (double) Choice of which parameters to work on, since depending on the task, not all SBtab parameters are worked on. k indices correspond to the parameters in the SBtab and numbers up to i correspond to the parameters in the work set. This is the set that actually gets used for diagnostics, optimization, and sensitivity analyis.

$$stg.partest_k = \begin{bmatrix} 1_{k_1} & 2_{k_2} & ... & i_{k_{end}} \end{bmatrix}$$

  In our example model parameter 216 from the SBtab is parameter number 1 of the work set, parameter 217 from the SBtab is parameter number 2 of the work set, and successively.

$$stg.partest_{[216:227]} = \begin{bmatrix} 1_{216} & 2_{217} & ... & 6_{221} & 1_{222} & 2_{223} & ... & 6_{227} \end{bmatrix}$$

- **stg.pat** - (double) Index(j) of the parameter set to work on
- **stg.pa** - (double) All the parameter sets

$$stg.pa = \begin{bmatrix} x_{1,1} & x_{2,1} & ... & x_{i,1} \\ x_{1,2} & x_{2,2} & ... & x_{i,2} \\ ... & ... & ... & ... \\ x_{1,j} & x_{2,j} & ... & x_{i,j} \end{bmatrix}$$

- **stg.bestpa** - (double) Best parameter set found so far during optimization

$$stg.bestx = \begin{bmatrix} bestx_1 & bestx_2 & ... & bestx_i \end{bmatrix}$$

  - x = Parameters being worked on
  - i = Index of Parameters being worked on
  - k = Index of the parameters in SBtab
  - j = Index of the Parameter set to work on

Plots

Default settings code

```
% True or false to decide whether to plot results (Plots)
stg.plot = true;

% True or false to decide whether to use long names in the title of the
% outputs plots in f_plot_outputs.m (Plot outputs long names)
stg.plotoln = true;
```

Example settings code

```
% True or false to decide whether to plot results (Plots)
stg.plot = true;

% True or false to decide whether to use long names in the title of the
% outputs plots in f_plot_outputs.m (Plot outputs long names)
stg.plotnames = true;
```

- **stg.plot** - (logical) Decide whether to plot results
- **stg.plotoln** - (logical) Decide whether to use long names in the title of the output plots in f_plot_outputs.m

Global Sensitivity Analysis (GSA)

Default settings code

```
% Number of samples to use in SA (Sensitivity analysis number of samples)
stg.sansamples = 100;

% True or false to decide whether to subtract the mean before calculating
% SI and SIT (Sensitivity analysis subtract mean)
stg.sasubmean = true;

% Choose the way you want to obtain the samples of the parameters for
% performing the SA; 0 Log uniform distribution truncated at the parameter
% bounds 1 Log normal distribution with mu as the best value for a
% parameter and sigma as stg.sasamplesigma truncated at the parameter
% bounds 2 same as 1 without truncation 3 Log normal distribution centered
% at the mean of the parameter bounds and sigma as stg.sasamplesigma
% truncated at the parameter bounds 4 same as 3 without truncation.
% (Sensitivity analysis sampling mode)
stg.sasamplemode = 2;

% Sigma for creating the normal distribution of parameters to perform
% sensitivity analysis (Sensitivity analysis sampling sigma)
stg.sasamplesigma = 0.1;
```

Example settings code

```
% Number of samples to use in SA (Sensitivity analysis number of samples)
stg.sansamples = 1000;

% True or false to decide whether to subtract the mean before calculating
% SI and SIT (Sensitivity analysis subtract mean)
stg.sasubmean = true;

% Choose the way you want to obtain the samples of the parameters for
% performing the SA; 0 Log uniform distribution truncated at the parameter
% bounds 1 Log normal distribution with mu as the best value for a
% parameter and sigma as stg.sasamplesigma truncated at the parameter
% bounds 2 same as 1 without truncation 3 Log normal distribution centered
% at the mean of the parameter bounds and sigma as stg.sasamplesigma
% truncated at the parameter bounds 4 same as 3 without truncation.
% (Sensitivity analysis sampling mode)
stg.sasamplemode = 2;

% Sigma for creating the normal distribution of parameters to perform
% sensitivity analysis (Sensitivity analysis sampling sigma)
stg.sasamplesigma = 0.1;

stg.gsabootstrapsize = ceil(sqrt(stg.sansamples));
```

- **stg.sansamples** - (double) Number of samples to use in GSA (in total (2+npars)*sansamples simulations will be performed, where npars are the number of parameters).
- **stg.sasubmean** - (logical) Decide whether to subtract mean before calculating *SI* and *STI*, see Halnes et al 2009.

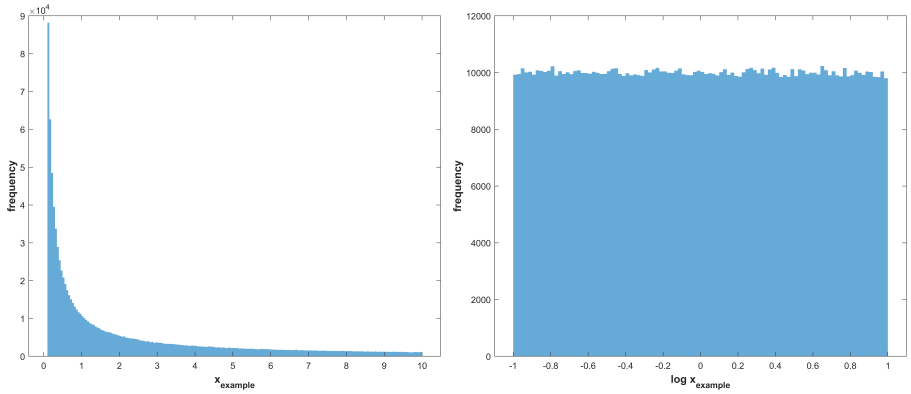
- **stg.sasamplemode** - (double) Choose the way you want to obtain the samples of the parameters for performing the GSA;

  0. Reciprocal (log uniform) distribution

     $X_i \sim Reciprocal(a_i, b_i)$

     - $i =$ Parameter index
     - $a_i = stg.lb_i$
     - $b_i = stg.ub_i$

     Example distribution with $a = -1, b = 1$

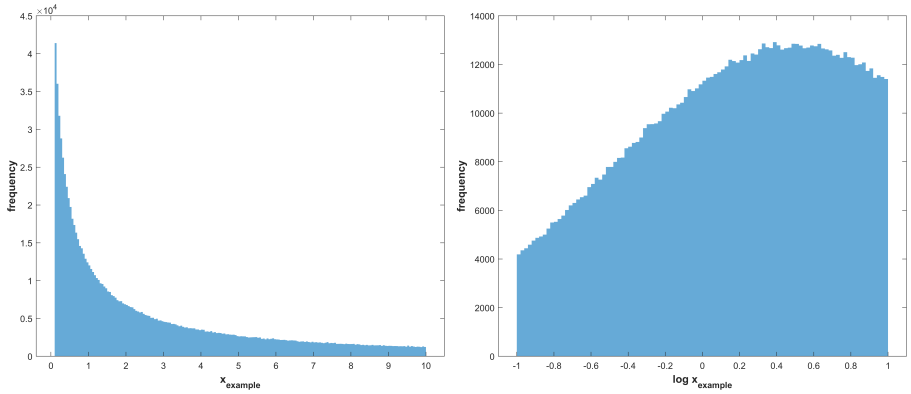


  1. Log normal distribution with corresponding to the best value for a parameter, as recieved from the optimization, and as *stg.sasamplesigma* truncated at the parameter bounds

     $X_i \sim TruncatedLogNormal(_i, , a_i, b_i)$

     - $i =$ Parameter index
     - $_i = bestx_i$
     - $= stg.sasamplesigma$
     - $a_i = stg.lb_i$
     - $b_i = stg.ub_i$

     Example distribution with $= 0.5, = 1, a = -1, b = 1$



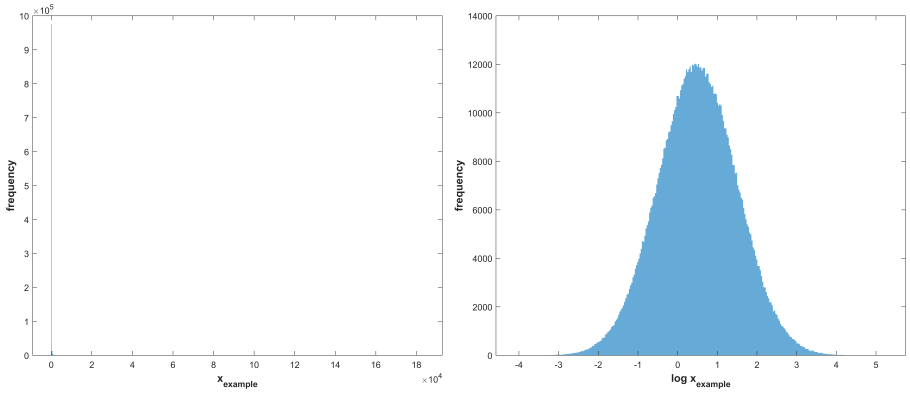
  2. same as 1 without truncation

     $X_i \sim LogNormal(,)$

     - $i =$ Parameter index

- $_i = bestx_i$
- $= stg.sasamplesigma$

Example distribution with $= 0.5, = 1$

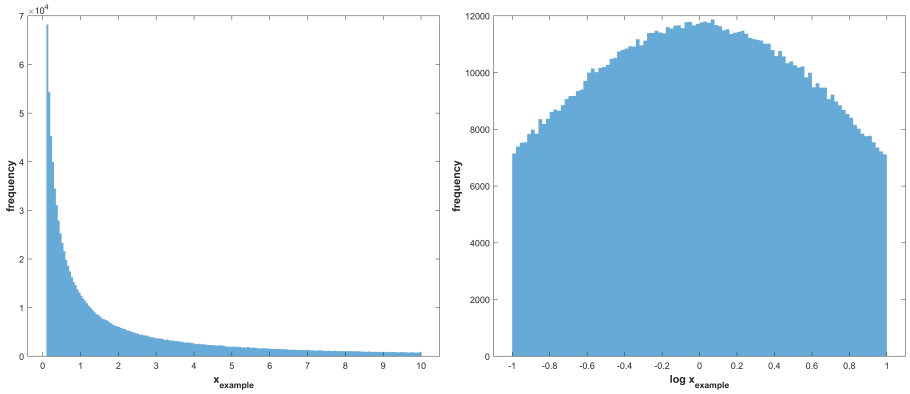


3. Log normal distribution with corresponding to the mean of the parameter bounds and as *stg.sasamplesigma* but truncated at the parameter bounds

   $X_i \sim TruncatedLogNormal(_i, , a_i, b_i)$

   - $i =$ Parameter index
   - $_i = \frac{stg.lb_i + (stg.ub_i - stg.lb_i)}{2}$
   - $= stg.sasamplesigma$
   - $a_i = stg.lb_i$
   - $b_i = stg.ub_i$

   Example distribution with $= \frac{a+(b-a)}{2}, = 1, a = -1, b = 1$

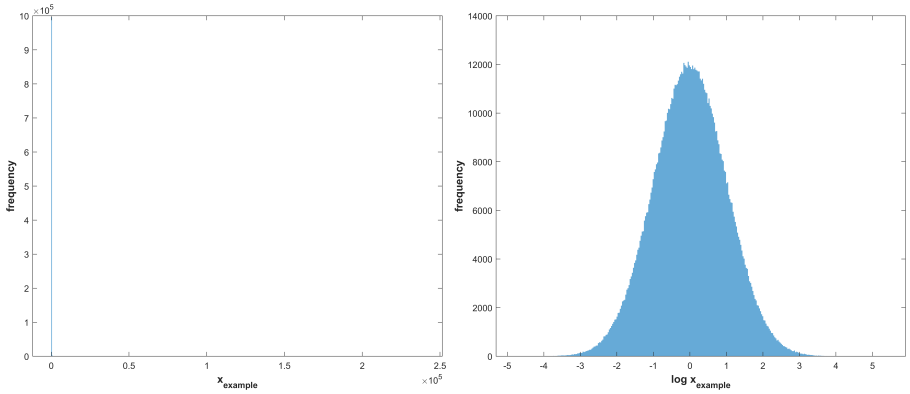


4. same as 3 without truncation.

   $X_i \sim LogNormal(mu_i,)$

   - $i =$ Parameter index
   - $_i = \frac{stg.lb_i + (stg.ub_i - stg.lb_i)}{2}$
   - $= stg.sasamplesigma$

   Example distribution with $= \frac{a+(b-a)}{2}, = 1, a = -1, b = 1$



- **stg.sasamplesigma** - (double) for creating the normal distribution of parameters to perform sensitivity analysis

Optimization

Default settings code

```
%  Time for the optimization in seconds (fmincon does not respect this
% time!!) (Optimization time)
stg.optt = 60*5;

% Population size for the algorithms that use populations (Population size)
stg.popsize = 144;

% optimization start method, choose between: 1 Random starting point or
% group of starting points inside the bounds 2 Random starting point or
% group of starting points near the best point (Optimization start method)
stg.osm = 1;

% Distance from best parameter array to be used in stg.osm method 2
% (Distance from best parameter array)
stg.dbs = 0.1;

% True or false to decide whether to use Multistart (Multistart)
stg.mst = false;

% Multistart size
stg.msts = 1;

% True or false to decide whether to display Plots (Plots doesn't work if
% using multicore) (Optimization plots)
stg.optplots = true;

% True or false to decide whether to run fmincon (no gradient so this
% doesn't work very well, no max time!!)
stg.fmincon = false;

% Options for fmincon (fmincon options)
stg.fm_options = optimoptions('fmincon',...
    'Algorithm','interior-point',...
    'MaxIterations',2,'OptimalityTolerance',0,...
```

(continues on next page)

(continued from previous page)

```
    'StepTolerance',1e-6,'FiniteDifferenceType','central',...
    'MaxFunctionEvaluations',10000);

% True or false to decide whether to run simulated annealing (Simulated
% annealing)
stg.sa = false;

% Options for simulated annealing (Simulated annealing options)
stg.sa_options = optimoptions(@simulannealbnd, ...
    'MaxTime',stg.optt,...
    'ReannealInterval',40);

% True or false to decide whether to run Pattern search (Pattern search)
stg.psearch = false;

% Options for Pattern search (Pattern search options)
stg.psearch_options = optimoptions(@patternsearch,...
    'MaxTime',stg.optt,'UseParallel',stg.optmc,...
    'UseCompletePoll',true,'UseCompleteSearch',true,...
    'MaxMeshSize',2,'MaxFunctionEvaluations',2000);

% True or false to decide whether to run Genetic algorithm (Genetic
% algorithm)
stg.ga = false;

% Options for Genetic algorithm (Genetic algorithm options)
stg.ga_options = optimoptions(@ga,'MaxGenerations',200,...
    'MaxTime',stg.optt,'UseParallel',stg.optmc,...
    'PopulationSize',stg.popsize,...
    'MutationFcn','mutationadaptfeasible');

% True or false to decide whether to run Particle swarm (Particle swarm)
stg.pswarm = false;

% Options for Particle swarm (Particle swarm options)
stg.pswarm_options = optimoptions('particleswarm',...
    'MaxTime',stg.optt,'UseParallel',stg.optmc,'MaxIterations',200,...
    'SwarmSize',stg.popsize);

% True or false to decide whether to run Surrogate optimization (Surrogate
% optimization)
stg.sopt = false;

% Options for Surrogate optimization (Surrogate optimization options)
stg.sopt_options = optimoptions('surrogateopt',...
    'MaxTime',stg.optt,'UseParallel',stg.optmc,...
    'MaxFunctionEvaluations',5000,...
    'MinSampleDistance',0.2,'MinSurrogatePoints',32*2+1);
end
```

Example settings code

```
%  Time for the optimization in seconds (fmincon does not respect this
% time!!) (Optimization time)
stg.optt = 60*1;

% Population size for the algorithms that use populations (Population size)
stg.popsize = 10;

% optimization start method, choose between: 1 Random starting point or
% group of starting points inside the bounds 2 Random starting point or
% group of starting points near the best point (Optimization start method)
stg.osm = 1;

% Distance from best parameter array to be used in stg.osm method 2
% (Distance from best parameter array)
stg.dbs = 0.1;

% True or false to decide whether to use Multistart (Multistart)
stg.mst = false;

% Multistart size
stg.msts = 1;

% True or false to decide whether to display Plots (Plots doesn't work if
% using multicore) (Optimization plots)
stg.optplots = true;

% True or false to decide whether to run fmincon (no gradient so this
% doesn't work very well, no max time!!)
stg.fmincon = false;

% Options for fmincon (fmincon options)
stg.fm_options = optimoptions('fmincon',...
    'UseParallel',stg.optmc,...
    'Algorithm','interior-point',...
    'MaxIterations',2,'OptimalityTolerance',0,...
    'StepTolerance',1e-6,'FiniteDifferenceType','central',...
    'MaxFunctionEvaluations',10000);

% True or false to decide whether to run simulated annealing (Simulated
% annealing)
stg.sa = false;

% Options for simulated annealing (Simulated annealing options)
stg.sa_options = optimoptions(@simulannealbnd, ...
    'MaxTime',stg.optt,...
    'InitialTemperature',...
    ones(1,stg.parnum)*2,'ReannealInterval',40);

% True or false to decide whether to run Pattern search (Pattern search)
stg.psearch = false;

% Options for Pattern search (Pattern search options)
stg.psearch_options = optimoptions(@patternsearch,...
```

(continues on next page)

(continued from previous page)

```
    'MaxTime',stg.optt,'UseParallel',stg.optmc,...
    'UseCompletePoll',true,'UseCompleteSearch',true,...
    'MaxMeshSize',2,'MaxFunctionEvaluations',2000);

% True or false to decide whether to run Genetic algorithm (Genetic
% algorithm)
stg.ga = true;

% Options for Genetic algorithm (Genetic algorithm options)
stg.ga_options = optimoptions(@ga,'MaxGenerations',200,...
    'MaxTime',stg.optt,'UseParallel',stg.optmc,...
    'PopulationSize',stg.popsize,...
    'MutationFcn','mutationadaptfeasible','Display','diagnose');

% True or false to decide whether to run Particle swarm (Particle swarm)
stg.pswarm = false;

% Options for Particle swarm (Particle swarm options)
stg.pswarm_options = optimoptions('particleswarm',...
    'MaxTime',stg.optt,'UseParallel',stg.optmc,...
    'SwarmSize',stg.popsize);

% True or false to decide whether to run Surrogate optimization (Surrogate
% optimization)
stg.sopt = false;

% Options for Surrogate optimization (Surrogate optimization options)
stg.sopt_options = optimoptions('surrogateopt',...
    'MaxTime',stg.optt,'UseParallel',stg.optmc,...
    'MaxFunctionEvaluations',5000,...
    'MinSampleDistance',0.2,'MinSurrogatePoints',32*2+1);
end
```

- **stg.optt** - (double) Time for the optimization in seconds (fmincon does not respect this time!!)
- **stg.popsize** - (double) Population size (for the algorithms that use populations)
- **stg.osm** - (double) optimization start method, choose between
    1. Get a random starting parameter set or group of starting parameter sets inside the bounds
    2. Get a random starting parameter set or group of starting parameter sets near the best parameter set
- **stg.dbpa** - (double) Distance from best parameter set to be used in *stg.osm* method 2
- **stg.mst** - (logical) Decide whether to use one or multiple starting parameter sets for the optimization
- **stg.msts** - (double) Number of starting parameter sets for the optimizations
- **stg.optplots** - (logical) Decide whether to display optimiazation plots (They aren't ploted if running the code in multicore)
- **stg.fmincon** - (logical) Decide whether to run fmincon (no gradient in our models so this doesn't work very well, does not respect *time set for the optimization*!!)
- **stg.fm_options** - (optim.options.Fmincon) Options for fmincon
- **stg.sa** - (logical) Decide whether to run simulated annealing

- **stg.sa_options** - (optim.options.SimulannealbndOptions) Options for simulated annealing
- **stg.psearch** - (logical) Decide whether to run Pattern search
- **stg.psearch_options** - (optim.options.PatternsearchOptions) Options for Pattern search
- **stg.ga** - (logical) Decide whether to run Genetic algorithm
- **stg.ga_options** - (optim.options.GaOptions) Options for Genetic algorithm
- **stg.pswarm** - (logical) Decide whether to run Particle swarm
- **stg.pswarm_options** - (optim.options.Particleswarm) Options for Particle swarm
- **stg.sopt** - (logical) Decide whether to run Surrogate optimization
- **stg.sopt_options** - (optim.options.Surrogateopt) Options for Surrogate optimization

Automatically generated at Import

- **stg.expn** - (double) Total number of experiments stored in the SBtab
- **stg.outn** - (double) Total number of experimental outputs specified in the SBtab

References

Halnes, G., Ulfhielm, E., Ljunggren, E.E., Kotaleski, J.H. and Rospars, J.P., 2009. Modelling and sensitivity analysis of the reactions involving receptor, G-protein and effector in vertebrate olfactory receptor neurons. Journal of Computational Neuroscience, 27(3), p.471.

3.2.7 Results

Scoring and saved simulation output

Every time a simulation is run the simulated results are compared to the results provided and a score is calculated. Additionally the end point of the experimental output of all simulations is also stored. When performing the diagnostics function an MATLAB® representation of the entire run is also saved.

- **simd** - Simulation results (MATLAB® representation)
- **st** - Total score

To simplify representations the following correspondence has been used

$score_{i,j,k} = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{Y_{i,j,k}-y_{i,j,k}}{_{i,j,k}}\right)^2$

if *stg.useLog* = 0

$st(;Y,) = \sum_{k=1}^{l}\sum_{j=1}^{m} score_{i,j,k}$

if *stg.useLog* = 1

$st(;Y,) = \sum_{k=1}^{l}\sum_{j=1}^{m} log_{10}(score_{i,j,k})$

if *stg.useLog* = 2

$st(;Y,) = \sum_{k=1}^{l} log_{10}(\sum_{j=1}^{m} score_{i,j,k})$

if *stg.useLog* = 3

$st(;Y,) = log_{10}(\sum_{k=1}^{l} \sum_{j=1}^{m} score_{i,j,k})$

- **se** - Score of each experiment

  if *stg.useLog* = 0 or 3

$$se(;Y,) = \begin{bmatrix} \sum_{j=1}^{m} score_{i,j,1} \\ \sum_{j=1}^{m} score_{i,j,2} \\ ... \\ \sum_{j=1}^{m} score_{i,j,k} \end{bmatrix}$$

  if *stg.useLog* = 1

$$se(;Y,) = \begin{bmatrix} \sum_{j=1}^{m} log_{10}(score_{i,j,1}) \\ \sum_{j=1}^{m} log_{10}(score_{i,j,2}) \\ ... \\ \sum_{j=1}^{m} log_{10}(score_{i,j,k}) \end{bmatrix}$$

  if *stg.useLog* = 2

$$se(;Y,) = \begin{bmatrix} log_{10}(\sum_{j=1}^{m} score_{i,j,1}) \\ log_{10}(\sum_{j=1}^{m} score_{i,j,2}) \\ ... \\ log_{10}(\sum_{j=1}^{m} score_{i,j,k}) \end{bmatrix}$$

- **sd** - Score of each experimental outputs in all experiments

  if *stg.useLog* = 0,2, or 3

$$sd(;Y,) = \begin{bmatrix} score_{i,1,1} & score_{i,2,1} & ... & score_{i,j,1} \\ score_{i,1,2} & score_{i,2,2} & ... & score_{i,j,2} \\ ... & ... & ... & ... \\ score_{i,1,k} & score_{i,2,k} & ... & score_{i,j,k} \end{bmatrix}$$

  if *stg.useLog* = 1

$$sd(;Y,) = \begin{bmatrix} log_{10}(score_{i,1,1}) & log_{10}(score_{i,2,1}) & ... & log_{10}(score_{i,j,1}) \\ log_{10}(score_{i,1,2}) & log_{10}(score_{i,2,2}) & ... & log_{10}(score_{i,j,2}) \\ ... & ... & ... & ... \\ log_{10}(score_{i,1,k}) & log_{10}(score_{i,2,k}) & ... & log_{10}(score_{i,j,k}) \end{bmatrix}$$

- **xfinal** - Value of each experimental outputs at the end of the simulation

$$xfinal(;Y,) = \begin{bmatrix} y_{n,1,1} & y_{n,2,1} & ... & y_{n,j,1} \\ y_{n,1,2} & y_{n,2,2} & ... & y_{n,j,2} \\ ... & ... & ... & ... \\ y_{n,1,k} & y_{n,2,k} & ... & y_{n,j,k} \end{bmatrix}$$

  - $F =$ Objective function for Particle Swarm optimization
  - $Y =$ Data provided for fitting
  - $y =$ Simulation results of the updated model under parameterization

- $=$ New parameterization for y
- $=$ Allowed mismatch between the two simulation results, analogous to the standard deviation of a Gaussian noise model in data fitting
- $n/i =$ Number/index of points in a given experimental output
- $m/j =$ Number/index of experimental outputs
- $l/k =$ Number/index of experiments

Diagnostics

When running the diagnostics a struct gets created that stores all the *oputputs* of the *f_sim_score function.*

- **rst.diag.simd** - Simulation results (MATLAB® representation)
- **rst.diag.st** - Total score
- **rst.diag.se** - Score per experiment
- **rst.diag.sd** - Score per experimental outputs in all experiments
- **rst.diag.xfinal** - x value of all the species being tested at the end of the simulation

Optimization

- **rst.opt.name** - Name of optimizer that was used
- **rst.opt.x** - Best parameter set found by the optimization
- **rst.opt.fval** - Score for that best parameter set
- **rst.opt.exitflag** - Diagnostics to see how the optimization went
- **rst.opt.output** - Diagnostics to see how the optimization went

Sensitivity Analysis

The calculations performed to obtain these sensitivities where performed according to the equations described in Halnes et al 2009.

- **rst.SA.M1** - Matrix with $(r * k)$ random numbers within the lower and upper bound ranges set for each parameter

$$M_1 = \begin{bmatrix} x_1^{(1)} & x_2^{(1)} & ... & x_k^{(1)} \\ x_1^{(2)} & x_2^{(2)} & ... & x_k^{(2)} \\ ... & ... & ... & ... \\ x_1^{(r)} & x_2^{(r)} & ... & x_k^{(r)} \end{bmatrix}$$

  - $x =$ Parameters
  - $k =$ Total number of parameters (*stg.parnum*)
  - $r =$ Total number of Samples (*stg.sansamples*)

- **rst.SA.M2** - Same as *rst.SA.M1* but different random initialization

$$M_2 = \begin{bmatrix} x_1^{(1')} & x_2^{(1')} & ... & x_k^{(1')} \\ x_1^{(2')} & x_2^{(2')} & ... & x_k^{(2')} \\ ... & ... & ... & ... \\ x_1^{(r')} & x_2^{(r')} & ... & x_k^{(r')} \end{bmatrix}$$

  - $x =$ Parameters
  - $k =$ Total number of parameters (*stg.parnum*)
  - $r =$ Total number of Samples (*stg.sansamples*)

- **rst.SA.N** - Matrix of size $(r * k * k)$ with columns exchanged between M1 and M2 as follows:

$$N_i = \begin{bmatrix} x_1^{(1')} & x_2^{(1')} & ... & x_i^{(1)} & ... & x_k^{(1')} \\ x_1^{(2')} & x_2^{(2')} & ... & x_i^{(2)} & ... & x_k^{(2')} \\ ... & ... & ... & ... & ... & ... \\ x_1^{(r')} & x_2^{(r')} & ... & x_i^{(r)} & ... & x_k^{(r')} \end{bmatrix}$$

  - $x =$ Parameters
  - $k =$ Total number of parameters (*stg.parnum*)
  - $r =$ Total number of Samples (*stg.sansamples*)
  - $i =$ Index of each parameter

- **rst.SA.fM1** -

$$fM_1 = \begin{bmatrix} f(M_1^{(1)}) \\ f(M_1^{(2)}) \\ ... \\ f(M_1^{(r)}) \end{bmatrix} = \begin{bmatrix} f(x_1^{(1)} & x_2^{(1)} & ... & x_k^{(1)}) \\ f(x_1^{(2)} & x_2^{(2)} & ... & x_k^{(2)}) \\ ... & ... & ... & ... \\ f(x_1^{(r)} & x_2^{(r)} & ... & x_k^{(r)}) \end{bmatrix}$$

  - $k =$ Total number of parameters (*stg.parnum*)
  - $r =$ Total number of Samples (*stg.sansamples*)

- **rst.SA.fM2** -

$$fM_2 = \begin{bmatrix} f(M_2^{(1')}) \\ f(M_2^{(2')}) \\ \dots \\ f(M_2^{(r')}) \end{bmatrix} = \begin{bmatrix} f(x_1^{(1')} & x_2^{(1')} & \dots & x_k^{(1')}) \\ f(x_1^{(2')} & x_2^{(2')} & \dots & x_k^{(2')}) \\ \dots & \dots & \dots & \dots \\ f(x_1^{(r')} & x_2^{(r')} & \dots & x_k^{(r')}) \end{bmatrix}$$

  - $k =$ Total number of parameters (*stg.parnum*)
  - $r =$ Total number of Samples (*stg.sansamples*)

- **rst.SA.fN** -

$$fN_i = \begin{bmatrix} f(N_i^{(1)}) \\ f(N_i^{(2)}) \\ \dots \\ f(N_i^{(r)}) \end{bmatrix} = \begin{bmatrix} f(x_1^{(1')} & x_2^{(1')} & \dots & x_i^{(1)} & \dots & x_k^{(1')}) \\ f(x_1^{(2')} & x_2^{(2')} & \dots & x_i^{(2)} & \dots & x_k^{(2')}) \\ \dots & \dots & \dots & \dots & \dots & \dots \\ f(x_1^{(r')} & x_2^{(r')} & \dots & x_i^{(r)} & \dots & x_k^{(r')}) \end{bmatrix}$$

  - $k =$ Total number of parameters (*stg.parnum*)
  - $r =$ Total number of Samples (*stg.sansamples*)
  - $i =$ Index of each parameter

- **rst.SA.SI** - First order effects

$S_i = \frac{V_i(E_{-i}(Y|_i))}{V(Y)} = \frac{U_i - E^2(Y)}{V(Y)}$

$U_i = \frac{1}{n-1} \sum_{r=1}^{n} f(M_1^r) f(N_i^r)$

$E^2(Y) = \frac{1}{n} \sum_{r=1}^{n} f(M_1^r) f(M_2^r)$

$V(Y) = \frac{1}{n-1} f^2(M_1^r) - E^2(Y)$

  - $V =$ Variance
  - $E(...|...) =$ Conditional expected value
  - $=$ Parameters of the model
  - $Y =$ Scalar output from the model
  - $n =$ Total number of Samples (*stg.sansamples*)
  - $r =$ Index of the Samples
  - $i =$ Index of each parameter

- **rst.SA.STI** - Total order effects

$S_{Ti} = \frac{V(Y) - V_i(E_i(Y|_i))}{V(Y)} = 1 - \frac{U_{-i} - E^2(Y)}{V_T(Y)}$

$U_{-i} = \frac{1}{n-1} \sum_{r=1}^{n} f(M_2^r) f(N_i^r)$

$E^2(Y) = \frac{1}{n} \sum_{r=1}^{n} f(M_1^r) f(M_2^r)$

$V_T(Y) = \frac{1}{n-1} f^2(M_2^r) - E^2(Y)$

  - $V =$ Variance
  - $E(...|...) =$ Conditional expected value
  - $=$ Parameters of the model
  - $Y =$ Scalar output from the model

- n = Total number of Samples (*stg.sansamples*)
- r = Index of the Samples
- i = Index of each parameter

References

Halnes, G., Ulfhielm, E., Ljunggren, E.E., Kotaleski, J.H. and Rospars, J.P., 2009. Modelling and sensitivity analysis of the reactions involving receptor, G-protein and effector in vertebrate olfactory receptor neurons. Journal of Computational Neuroscience, 27(3), p.471.

3.2.8 Model files and folders

Here we have the file hierarchy of the models used in MATLAB® portion of our workflow.
Some of these folders and files need to be user generated before running the code and some are automatic generated at runtimne, in the list below the former are identified as (u-gen) and the later as (a-gen)
When importing one of our provided models the user should place the model folder inside the "Subcellular_workflow/Matlab/Model/" all this folders and files are relative to this folder.

- **"Model Folder name"/ (u-gen)**

  Folder containing all the model files the model, we recomend using the same name as the repository name for the models we provide but there is no restrictions.

  - "Source_sbtab_name".xlsx (u-gen)

    Contains the SBtab in .xlsx format.

  - **Matlab/** (u-gen)

    Folder containing all model files related to MATLAB®, the model is used in other softwares so here reside all the files that MATLAB® uses or generates

    * **Data/** (a-gen)

      Folder containing the model in severall diferent formats relevant for the analysis and files contaning model metadata such as experimental inputs and outputs

      * model_"model name".sbproj (a-gen)

        Contains the model derived from the SBtab in .sbproj (MATLAB® SimBiology®) format.

      * model_"model name".mat (a-gen)

        Contains the model derived from the SBtab in .mat format.

      * data_"model name".mat (a-gen)

        Contains data derived from the SBtab in a .mat format. This data is used to run the model taking into account all the inputs and outputs of the model.

      * model_"model name".xml (a-gen)

        Contains the model derived from the SBtab in .xml (SBML) format.

- Input_"model name".mat (a-gen)

  Contains input data derived from the SBtab in a .mat format for all the experimental inputs.

- SBtab_"model name".mat (a-gen)

  Contains the SBtab in .mat format.

- **Exp/** (a-gen)

  Contains a version of the model for each experiment contained in the SBtab. They include all the neccessary inputs and outputs to simulate the supplied experimental conditions.

  - Model_"model name"_i.mat (a-gen)

    Tailor made for the main run of the simulation.

  - Model_eq_"model name"_i.mat (a-gen)

    Tailor made for the equilibration step of the simulation.

  - Model_detail_"model name"_i.mat (a-gen)

    Tailor made for the main run of the simulation. The step size is reduced to generate better graphs

- **Input_functions/** (a-gen)

  Folder containing the functions that are used at run time to give the correct input to all experiments

  - "model name"_inputi_Ligand.mat (a-gen)

    These functions interpolate the input that is supposed to be given to the model at run time.

  - "model name"_input_creator.mat (a-gen)

    Creates the functions above for all experimental inputs.

- **Results/** (a-gen)

  Folder containing the results of the various possible Matlab analysis provided by our workflow

  - **"Analysis name"/** (a-gen)

    Each analysis output is stored in its own folder, depending on the analysis run the the results can be saved in either a "Diagnostics", "Optimization" or "Sensitivity Analysis". An "Examples" folder is also provided with analysis that were pre-run by us.

  - **"date"/** (a-gen)

    The date and time of when the analysis was run, this is auto generated when an user choses to run any alysis.

  - All_figures.fig (a-gen)

    All the plots generated by the analysis stored in a Matlab figure assembly

  - Analysis.mat (a-gen)

    All the data used as input to the analysis, saved as the SBtab and the setting fileconverted to a matlab structs called "sb" and "stg" respectively. And all the outputs generated by running the analysis saved also in a matlab struct called "*rst*"

  - "Figure name".png (a-gen)

    All the plots generated by the analysis stored individually as images

        * **Settings/** (u-gen)

          Folder containing the *settings file*

            · "Settings file name" (u-gen)

              Settings file of the model. A place for the user to define all the relevant properties of model simulation that are not stored in SBtab. These are usually things that need to change during optimizations or model development.

    – **tsv/** (a-gen)

      Folder containing the SBtab converted to .tsv files for each SBtab that as been run through one of our analysis.

        * **"model name"** (a-gen)

          Contains the SBtab in .tsv format.

Description of the general terms:

"Model Folder name" - Name of the folder containing the model files and folders. We recomend using the same name as the repository name for the models we provide but there is no restrictions.
"Source_sbtab_name" - Name of the SBtab provided by the user
"model name" - Name given to the model in all the automatic generated model files, chosen in the *settings file*
"Settings file name" - Name chosen by the user for the *settings file* of the model. By default we chose the same as the "model name".
"Analysis name" - Depending on the analysis run the the results can be saved in either a "*Diagnostics*", "*Optimization*" or "*Sensitivity Analysis*" folder. An "Examples" folder is also provided with analysis that were pre-run by us.
"date" - The date and time of when the analysis was run, this is auto generated when an user choses to run any analysis.
"Figure name" - Catch all for all description for the names of all the plots that are generated by our analysis.

3.3 NEURON

Here we describe the usage of the MOD files for two of the provided examples (the third one does not use MOD files).

3.3.1 Nair et al. 2016 (D1 MSN subcellular cascade)

The model requires input in the form of dopamine and calcium. These need to be specified by the user:

1. Dopamine is set to be 20 nM in assign_calculated_values(). This line should be replaced to whatever the user needs for input. For example, it could be replaced with an expression for dopamine such as the one provided in this mod file, which makes a dopamine pulse with a double exponential shape. The line

DA = 20

should instead read

DA = DA_expression.

1. The same goes for the calcium input. Alternatively, calcium could be provided via the intracellular concentration of a calcium ion. For example, in this MOD file we use the intracellular calcium concentration due to influx from NMDA receptors, ca_nmdai, which is calculated through a mechanism for calcium accumulation provided in a

separate MOD file. Access to the ionic concentrations is provided by NEURON's "USEION" statement. In this case the variable ca_nmdai needs to be added to the ASSIGNED block.

———*IMPORTANT NOTE*———

When specifying the calcium input like this care needs to be taken to make sure the units used by NEURON and the units in the model exported to the MOD file match. As mentioned in the article, the models exported as MOD files could have different units for the parameter values from the default units used by NEURON. For instance, NEURON's default units for internal calculations of ionic concentrations are in millimolars (mM), but the model's parameters are expressed in nanomolars (nM). It is absolutely *paramount* to match units, i.e. use the correct scaling for, in this case, the variable ca_nmdai, to provide the model with the right quantity of calcium so that it runs properly:

Ca = ca_nmdai * (1e6)

3.3.2 Viswan et al. 2016 (EGF-stimulated MAPK cascade)

In this model we only use EGF as an input (only this input is used in Figure 7 in Viswan et al. 2018 which we reproduce; otherwise the model may have various other inputs, see the original paper for details [2]).

1. The input is a step in the concentration of EGF, given by an expression for a sharp sigmoidal function for EGF in assign_calculated_values():

EGF = EGF_level/(1+exp(-(t-EGF_start) * EGF_steepness))

1. The SBtab format of this model expresses the parameter values in seconds, whereas NEURON's default unit for time is milliseconds. The conversion script SBtab_to_vfgen provides automatic scaling of time units in SBtab to milliseconds. The concentration units are given in micromolar, and if some species needs to be coupled to a NEURON variable which is expressed in other units (such as NEURON's default millimolar units), the species or the NEURON variable need to be rescaled as in the example above.

3.3.3 References

[1] Nair, A.G., Bhalla, U.S. and Hellgren Kotaleski, J., 2016. Role of DARPP-32 and ARPP-21 in the emergence of temporal constraints on striatal calcium and dopamine integration. PLoS computational biology, 12(9), p.e1005080.

[2] Viswan, N.A., HarshaRani, G.V., Stefan, M.I. and Bhalla, U.S., 2018. FindSim: a framework for integrating neuronal data and signaling models. Frontiers in neuroinformatics, 12, p.38.

3.4 Subcellular application

Subcellular application (https://subcellular.humanbrainproject.eu/model/meta) provides a web interface for simulation of biomolecular networks expressed on bionetgen language (https://bionetgen.org/) using network free solver NFsim and reaction-diffusion stochastic systems solver STEPS (http://steps.sourceforge.net/STEPS/documentation.php) Models can be imported from an sbml file. In this repository we used two model examples to exemplify the usage of this tool.

3.4.1 BioNetGen translation of SBtab Nair_2016 model

The model was translated from SBtab model format to rule-based BioNetGen language for the simulation with STEPS and NFsim solvers embedded in the subcellular web app and with the RuleBender

Conversion steps

- Run ***convert_Nair_2016_from_SBTAB_to_SBML.R*** in RStudio to translate SBtab model to SBML. This step depends on SBtab to SBML converter
- Run ***convert_Nair_2016_from_SBML_to_BNGL.ipynb*** jupyter notebook to translate from SBML to BioNetGen language
- Import the resulted BioNetGen model ***Nair_2016_optimized_alternative.bngl*** to the **subcellular web app**. Add spine geometry ***.json***, ***.node***, ***.ele***, ***.face*** files and stimulation pattern ***stim_DA_complex.tsv***. See the **subcellular web app help** for details
- Simulate the final model ***Nair_2016_optimized_alternative.ebngl*** in the subcellular web app using **STEPS** or **NFsim** solvers

Files and folders

Nair_2016_optimized_alternative.ebngl - extended BNGL model corresponding to the ***optimized Nair 2016*** SBtab model with added geometry and stimulation patterns. Can be imported and simulated in the **subcellular web app**

SBTAB_Nair_2016 - the folder with the ***optimized Nair 2016*** SBtab model tsv tables.

Nair_2016_optimized.xml - SBML model translated from the ***optimized Nair 2016*** model by ***convert_Nair_2016_from_SBTAB_to_SBML.R*** script based on SBtab to SBML converter

Nair_2016_optimized_alternative.bngl - BioNetGen model obtained from ***Nair_2016_optimized.xml*** by ***convert_Nair_2016_from_SBML_to_BNGL.ipynb*** jupyter notebook which is based on ***sbml_to_bngl.py*** conversion tool

spine.ele, ***spine.face***, ***spine.node***, ***spine.json*** - these files specify TetGen meshes and model geometry needed for the subcellular web app *Geometry* section and STEPS solver (see the **subcellular web app help** for details)

stim_DA_complex.tsv, ***stim_noDA_complex.tsv*** - these files specify the stimulation pattern in *Simulations* section of the subcellular web app (corresponds to the experiments E0 - E9 of the SBtab model).

SBtabVFGEN-master - the folder containing a copy of SBtab to SBML converter

sbml_to_bngl.py - the python tool for conversion of SBML models to BioNetGen language.

3.4.2 BioNetGen translation of SBtab Viswan_2018 model

The model was translated from SBtab model format to rule-based BioNetGen language for the simulation with STEPS and NFsim solvers embedded in the subcellular web app and with the RuleBender

Conversion steps

- Run ***convert_Viswan_2018_for_STEPS_optimised_from_SBTAB_to_SBML.R*** in RStudio to translate SBtab model to SBML. This step depends on SBtab to SBML converter and LibSBML
- Run ***convert_Viswan_2018_for_STEPS_optimised_from_SBML_to_BNGL.ipynb*** jupyter notebook to translate from SBML to BioNetGen language. This step depends on ***sbml_to_bngl.py*** and on LibSBML or pySB
- Import the BioNetGen model (***SBTAB_Viswan_2018_alternative.bngl***) to the **subcellular web app**. Add spine geometry (***.json***, ***.node***, ***.ele***, ***.face*** files) and stimulation pattern (***stim_E0.tsv***). See the **subcellular web app help** for details
- Simulate final model (***SBTAB_Viswan_2018_alternative.ebngl***) in the subcellular web app using **STEPS** or **NFsim** solvers
- Simulate the BioNetGen model with the RuleBender

Files and folders

SBTAB_Viswan_2018_alternative.ebngl - extended BNGL model corresponding to the ***Viswan_2018_for_STEPS_optimised.xlsx*** SBTAB model with added geometry and stimulation patterns. Can be imported and simulated in the **subcellular web app**

Viswan_2018_for_STEPS.xlsx - SBtab model equivalent to the original Viswan_2018 model. It was obtained from ***Viswan_2018.xlsx*** by the modification of the model features incompatible with BNGL.

Viswan_2018_for_STEPS_optimised.xlsx - SBtab model equivalent to the optimized Viswan_2018 model. It was obtained from ***Viswan_2018_optimized.xlsx*** by the modification of the model features incompatible with BNGL.

SBTAB_Viswan_2018.xml - SBML model translated from ***Viswan_2018_for_STEPS_optimised.xlsx*** model by ***convert_Viswan_2018_for_STEPS_optimised_from_SBTAB_to_SBML.R*** script based on SBtab to SBML converter

SBTAB_Viswan_2018_alternative.bngl - BioNetGen model obtained from ***SBTAB_Viswan_2018.xml*** by ***convert_Viswan_2018_for_STEPS_optimised_from_SBML_to_BNGL.ipynb*** jupyter notebook which is based on ***sbml_to_bngl.py*** conversion tool

cell.ele, ***cell.face***, ***cell.node***, ***cell.json*** - these files specify TetGen meshes and model geometry needed for the subcellular web app *Geometry* section and STEPS solver (see the **subcellular web app help** for details)

stim_E0.tsv, ***stim_E1.tsv*** - these files specify the stimulation pattern in *Simulations* section of the subcellular web app (corresponds to the experiments E0 and E1 of the SBtab model).

Viswan_2018_alternative_RuleBender.bngl - BNGL model corresponding to the ***Viswan_2018_for_STEPS_optimised.xlsx*** SBtab model with additional section specifying stimulation and BioNetGen solver. Can be imported and simulated in the RuleBender

SBtabVFGEN-master - the folder containing copy of SBtab to SBML converter

SBTAB_Viswan_2018_for_STEPS_optimised - the folder containing **tsv** tables of ***Viswan_2018_for_STEPS_optimised.xlsx***

sbml_to_bngl.py - the python tool for conversion of SBML models to BioNetGen language.

3.4.3 Conversion of SBML to BioNetGen language

The conversion is implemented in ***sbml_to_bngl.py*** python module. Two approaches are supported by *sbml_to_bngl.transform()* function:

- if *converter='pysb'* - the converter based on the Atomizer implemented in pysb.importers.sbml.sbml_translator() function within pySB framework will be used. The Atomizer will try to modify the set of model molecules and reactions to convert them from reaction network to rule-based BioNetGen format.
- *if converter='plain'* - a libsbml based converter for sbml level 2, version 4 will be used. This converter produces a bngl approximation to reaction network format of a model. It is assumed that sbml models were obtained by exporting a MATLAB simbiology model to sbml, or by translation of SBTAB model by SBtab to SBML converter.

Models expressed by SBML and SBTAB often are not fully compatible with BNGL. Additional model adaptation steps are required in this case to obtain a working BNGL model. These steps will be partially automatized by *sbml_to_bngl.transform()* function if *adapt_steps* argument dictionary *'list_of_steps'* is nonempty.

The adaptation steps include:

1. The STEPS and NFsim solvers require different units for species quantities an kinetic rates. An adapted BNGL model provides modifed expressions for all species concentrations and kinetic rates and provides an easy way for units changing by specification of auxilary bngl model parameters: *Na* and *V_comparment_name*. These parameters should be selected to: *Na=6.022e23* and *V_comparment_name* = volume of corresponding compartment in liters for NFsim and to: *Na=1* and *V_comparment_name=1* for STEPS.
2. Species with fixed concentrations are not supported by NFsim solver. The BNGL model adaptation will modify model reactions such that a fixed species concentration became a model parameter. This parameter can be used for the clamping of species concentration or for the stimulation pattern application
3. If SBML to BNGL converter implemented in Atomizer is selected then additional transformation steps include renaming of duplicated molecule sites and reparing incorrect molecule names and kinetic rate transformations
4. In case when MATLAB simbiology is used for SBML model creation, adapt_steps will repare incorrect molecule and parameter names
5. Compartmental model of BNGL assumes tree structure of the set of model comartments. This assumption is often incompatible with mesh geometries supported by STEPS. The model adaption steps can produce bngl models with flexible compartmental structure
6. There is a number of incompatibilities between SBTAB and BNGL which still require manual correction. These include concentrations fixed to an expression, functional expressions for reaction rates in case of STEPS solver, some nonstandard types of reaction kinetic functions etc. The adapt_steps detects the cases of known incompatibilities and produces corresponding warning messages

3.5 Conversion tools

For this workflow we have developed some conversion tools to facilitate model developement.

The MATLAB® code takes the SBtab file in excel format and generates tab separeted file (.tsv) of this SBtab, an SBML file (.xml) of the model, and two MATLAB® versions of the model (.m and .sbproj). This conversion happens as a setup step of running any of the Matlab analysis, it might be added as a standalone option in the future.

SBtab (.xls,.xlsx) -> Matlab® model (.m .sbproj), SBtab (.tsv), Matlab® SBML (.xml)

We also have R code to perform other conversions in two external repositories:

- Code for fixing the SBML produced by MATLAB®

https://github.com/a-kramer/simbiology-sbml-fix

Matlab® SBML (.xml) -> SBML (.xml)

- A standalone SBtab to VFgen SBML and NEURON tool

  https://github.com/a-kramer/SBtabVFGEN

  SBtab (.tsv or .ods) -> VFGEN (.vf) + SBML (.xml) + Neuron (.mod)

3.6 Models

We have used multiple models to validate our software tools. Each model has its own repository. In these repositories you can find;

- The model in different formats relevant for the various tools that we provide
- "Matlab" folder with:
  - Settings file relevant to use the model with our MATLAB® tools
  - Some examples of the output provided by our MATLAB® tools after running an Analysis

  This is also the folder were all the run time ouptuts of the matlab analysis are stored.
- "Bionetgen and Steps" folder with relevant files for running the model in the subcellular aplication
- "Neuron" folder with relevant files for running the model in NEURON.

Note: Model_Fujita_2010 does not contain "Bionetgen and Steps" and "Neuron" folders as this model was not implemented in these softwares.

Links to the model repositories:

- https://github.com/jpgsantos/Model_Nair_2016
- https://github.com/jpgsantos/Model_Fujita_2010
- https://github.com/jpgsantos/Model_Viswan_2018

3.6.1 Fujita et al. 2010

Model by Fujita et al 2010 as one of the proposed benchmark models provided by Hass et al. 2019. The model represents epidermal growth factor (EGF)-dependent Akt pathway. Data from [Fig. 1b] with a corresponding EGF step input from [Fig. 1a] in Fujita et al. 2010 was used for estimation and Global Sensitity Analysis of 12 parameters. We performed parameter estimation starting from the model parameters provided in the publication, using a search space 5 order of magnitude above and below for each parameter.

- EGFR + EGF <=> EGF_EGFR
- pEGFR + Akt <=> pEGFR_Akt
- pEGFR_Akt -> pEGFR + pAkt
- pEGFR -> null
- pAkt + S6 <=> pAkt_S6
- pAkt_S6 -> pAkt + pS6
- pAkt -> Akt
- pS6 -> S6

- EGF_EGFR -> pEGFR
- EGFR -> null

https://github.com/jpgsantos/Model_Fujita_2010

Fujita, K.A., Toyoshima, Y., Uda, S., Ozaki, Y.I., Kubota, H. and Kuroda, S., 2010. Decoupling of receptor and downstream signals in the Akt pathway by its low-pass filter characteristics. Science Signaling, 3(132), pp.ra56-ra56.

Hass, H., Loos, C., Raimúndez-Álvarez, E., Timmer, J., Hasenauer, J. and Kreutz, C., 2019. Benchmark problems for dynamic modeling of intracellular processes. Bioinformatics, 35(17), pp.3073-3082.

3.6.2 Nair et al. 2016

We illustrate the Subcellular Workflow with a model depicting the emergence of the eligibility trace observed in reinforcement learning in striatal D1 medium spiny neurons (D1 MSN) (Nair et al. 2016). Here, an intracellular increase in calcium representing excitatory synaptic input leads to synaptic potentiation only when it is followed by a reinforcing dopamine input. These two signaling cascades, starting with a calcium train and a dopamine transient, are illustrated in Fig. 1a. The first pathway (depicted in blue) represents calcium-dependent activation of Ca2+/calmodulin-dependent protein kinase II (CaMKII), its autophophorylation, and the phosphorylation of a generic CaMKII substrate that represents long term potentiation (LTP). In the second pathway (species in red), dopamine initiates a G-protein dependent cascade which results in the phosphorylation of dopamine- and cAMP-regulated phosphoprotein, 32 kDa (DARPP-32) turning into an inhibitor of protein phosphatase 1 (PP1) that otherwise dephosphorylates CaMKII and its substrate. Substrate phosphorylation is maximal when the time window between the calcium and dopamine inputs is sufficiently short (input-interval constraint mediated by DARPP-32 via PP1 inhibition), and when intracellular calcium elevation is followed by dopamine (input-order constraint mediated by another phosphoprotein, the cyclic AMP-regulated phosphoprotein, 21 kDa (ARPP-21) that sequesters calcium/calmodulin if dopamine arrives before calcium).

CaMKII is autophosphorylated both in the cytosol and the post synaptic density (PSD) with a custom MATLAB rate function as described in Li et al. (2012). To run the model in different software, we substitute the autophosphorylation rate function with the same set of bimolecular reactions (simplified version of reactions from Pepke et al. 2010) for both compartments. The reactions represent the formation of a complex with two fully activated CaMKII monomers and a catalytic step in which one monomer phosphorylates the other. Schematics can be found in Fig. 1b along with the six new parameters. We estimated the parameters using simulated data from the published model with simulation setups in Fig. 1c depicting different timings of the dopamine input relative to the calcium input.

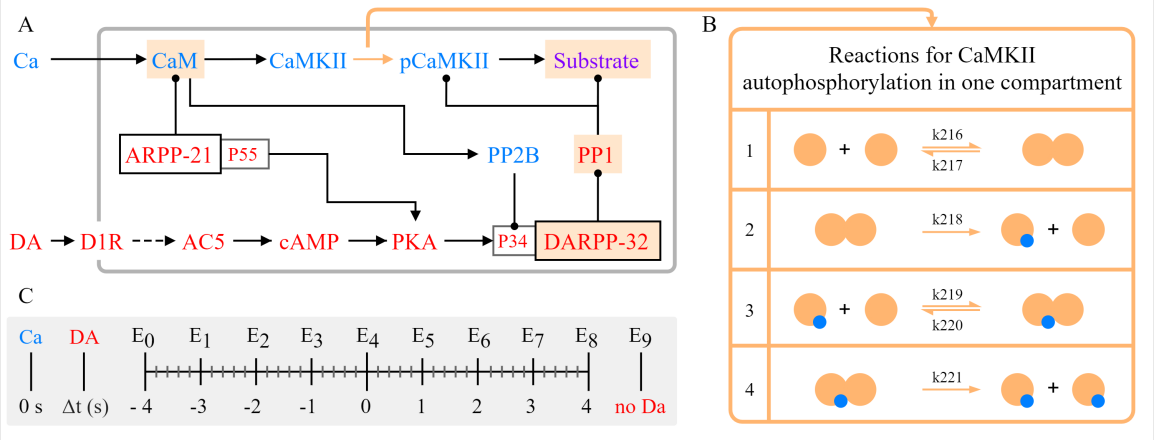


Figure 1. (A) Simplified schematics of the model. Species in the calcium cascade are depicted in blue, and species in the dopamine cascade are depicted red. Simulated time course data of the species with a beige background were used in parameter estimation. Figure adapted from Nair et al. (2016). (B) Illustration of the bimolecular reactions in CaMKII autophosphorylation with yellow circles representing activated CaMKII, and blue circles representing phosphate groups. Newly introduced parameters with their IDs in the updated model are shown above/below the arrows.

(C) Timing of the dopamine input (t ={-4,-3,-2,-1,0,1,2,3,4} corresponding to experiments E0-E8) relative to calcium increase (time 0), and a single experiment without dopamine (E9).

Four reactions representing autophosphorylation in one compartment.

Reactions in cytosol:

- CaMKII_CaM_Ca4 + CaMKII_CaM_Ca4 <=> CaMKII_CaM_Ca4_CaMKII_CaM_Ca4
- CaMKII_CaM_Ca4_CaMKII_CaM_Ca4 -> pCaMKII_CaM_Ca4 + CaMKII_CaM_Ca4
- pCaMKII_CaM_Ca4 + CaMKII_CaM_Ca4 <=> pCaMKII_CaM_Ca4_CaMKII_CaM_Ca4
- pCaMKII_CaM_Ca4_CaMKII_CaM_Ca4 -> pCaMKII_CaM_Ca4 + pCaMKII_CaM_Ca4

Reactions in PSD:

- CaMKII_CaM_Ca4_psd + CaMKII_CaM_Ca4_psd <=> CaMKII_CaM_Ca4_psd_CaMKII_CaM_Ca4_psd
- CaMKII_CaM_Ca4_psd_CaMKII_CaM_Ca4_psd -> pCaMKII_CaM_Ca4_psd + CaMKII_CaM_Ca4_psd
- pCaMKII_CaM_Ca4_psd + CaMKII_CaM_Ca4_psd <=> pCaMKII_CaM_Ca4_psd_CaMKII_CaM_Ca4_psd
- pCaMKII_CaM_Ca4_psd_CaMKII_CaM_Ca4_psd -> pCaMKII_CaM_Ca4_psd + pCaMKII_CaM_Ca4_psd

https://github.com/jpgsantos/Model_Nair_2016

Li, L., Stefan, M.I. and Le Novère, N., 2012. Calcium input frequency, duration and amplitude differentially modulate the relative activation of calcineurin and CaMKII. PloS one, 7(9), p.e43810.

Nair, A.G., Bhalla, U.S. and Hellgren Kotaleski, J., 2016. Role of DARPP-32 and ARPP-21 in the emergence of temporal constraints on striatal calcium and dopamine integration. PLoS computational biology, 12(9), p.e1005080.

Pepke, S., Kinzer-Ursem, T., Mihalas, S. and Kennedy, M.B., 2010. A dynamic model of interactions of Ca 2+, calmodulin, and catalytic subunits of Ca 2+/calmodulin-dependent protein kinase II. PLoS Comput Biol, 6(2), p.e1000675.

3.6.3 Viswan et al. 2018

Model from the FindSim framework by Viswan et al. 2018. The model represents an epidermal growth factor (EGF)-dependent mitogen-activated protein kinase (MAPK) signaling pathway (the green block from [Fig. 7a]) which measures MAPK phosphorylation. Here, two simulation experiments with EGF step inputs of different sizes are used to perform parameter estimation on 29 model parameters corresponding to reactions involved in MAPK phosphorylation (see below). For this we used activated MAPK curves in [Fig. 7b and 7c].

- GTP_Ras + craf_1_p <=> Raf_p_GTP_Ras
- GEF_p -> inact_GEF
- GTP_Ras -> GDP_Ras
- GAP_p -> GAP
- MAPK_p_p + craf_1_p <=> MAPK_p_p_feedback_cplx
- MAPK_p_p_feedback_cplx -> MAPK_p_p + craf_1_p_p
- MAPKK_p_p + MAPK <=> MAPKKtyr_cplx
- MAPKKtyr_cplx -> MAPKK_p_p + MAPK_p
- MAPKK_p_p + MAPK_p <=> MAPKKthr_cplx
- MAPKKthr_cplx -> MAPKK_p_p + MAPK_p_p
- MAPKK + Raf_p_GTP_Ras <=> Raf_p_GTP_Ras_1_cplx
- Raf_p_GTP_Ras_1_cplx -> MAPKK_p + Raf_p_GTP_Ras

- MAPKK_p + Raf_p_GTP_Ras <=> Raf_p_GTP_Ras_2_cplx
- Raf_p_GTP_Ras_2_cplx -> MAPKK_p_p + Raf_p_GTP_Ras
- inact_GEF + GDP_Ras <=> basal_GEF_activity_cplx
- basal_GEF_activity_cplx -> inact_GEF + GTP_Ras
- GEF_p + GDP_Ras <=> GEF_p_act_Ras_cplx
- GEF_p_act_Ras_cplx -> GEF_p + GTP_Ras
- GAP + GTP_Ras <=> GAP_inact_Ras_cplx
- GAP_inact_Ras_cplx -> GAP + GDP_Ras

https://github.com/jpgsantos/Model_Viswan_2018

Viswan, N.A., HarshaRani, G.V., Stefan, M.I. and Bhalla, U.S., 2018. FindSim: a framework for integrating neuronal data and signaling models. Frontiers in neuroinformatics, 12, p.38.

CHAPTER

FOUR

REFERENCES

Santos, J.P., Pajo, K., Trpevski, D., Stepaniuk, A., Eriksson, O., Nair, A.G., Keller, D., Kotaleski, J.H. and Kramer, A., 2020. A Modular Workflow for Model Building, Analysis, and Parameter Estimation in Systems Biology and Neuroscience. bioRxiv.

Lubitz, T., Hahn, J., Bergmann, F.T., Noor, E., Klipp, E. and Liebermeister, W., 2016. SBtab: a flexible table format for data exchange in systems biology. Bioinformatics, 32(16), pp.2559-2561.

Halnes, G., Ulfhielm, E., Ljunggren, E.E., Kotaleski, J.H. and Rospars, J.P., 2009. Modelling and sensitivity analysis of the reactions involving receptor, G-protein and effector in vertebrate olfactory receptor neurons. Journal of Computational Neuroscience, 27(3), p.471.

Nair, A.G., Bhalla, U.S. and Hellgren Kotaleski, J., 2016. Role of DARPP-32 and ARPP-21 in the emergence of temporal constraints on striatal calcium and dopamine integration. PLoS computational biology, 12(9), p.e1005080.